ELBUG

MONITOR-SOFTWARE

für den

ELEKTOR-SC/MP-MIKROCOMPUTER

Dr. H. HUSCHITT

Inhalt

		Seite
1.	Einführung	3
2.	Arbeiten mit dem ELBUG	4
3.	Listing ELBUG	10
4.	Flußdiagramme	28
5.	Erläuterungen zu den einzelnen Routinen	32

Verfasser: Dr. H. Huschitt, D-684 Lampertheim 5

Urheberrechtlich geschützt. Jedes Veräußern, Verleihen oder sonstiges Verbreiten dieser Dokumentation, auch auszugsweise, sowie die gewerbliche Nutzung des Programms bedürfen der Genehmigung des Verfassers.

Druck: HELD-DRUCK 6842 Bürstadt

November 1977

1. Einleitung

Um Mikroprozessoren zur besseren, flexibleren und preisgünstigeren Lösung von Aufgaben der Digital-Elektronik einsetzen zu können, benötigt man zunächst eine Einrichtung zum Entwickeln u. Austesten von Programmen - ein *Entwicklungs-System.*

Das 7-Segment-Hexadezimal-Ausbaustadium des ELEKTOR-SC/MP-Mikrocomputers zusammen mit der Betriebs-Software „ ELBUG " stellt bereits ein einfaches Entwicklungssystem dar. Die etwas bescheidene Architektur und Befehlsstruktur des SC/MP wird durch dieses Gesamt-System so erweitert (LIFO-Stack, Block-Umlade-Kommando, 8 Interrupt-Eingänge usw.), daß das Komfort-Niveau wesentlich teurerer uP's erreicht wird.

Die weiteren, in ELEKTOR geplanten Artikel (RAM-Erweiterungs-Karten, PROM-Programmier-Platine, ASCII-Keyboard und TV-Interface) vervollständigen das System soweit, daß die vorhandene SC/MP-residente Software(z.Zt. zwei Assembler-Versionen und BASIC-Interpreter) voll ausgenutzt werden kann, ohne teure Peripherien, wie Sichtgeräte, Drucker, Massenspeicher usw. zu benötigen.

Es hat sich als sinnvoll erwiesen, Programme für einen bestimmten Mikroprozessor auf einem Mikrocomputer, der eben diesen uP als CPU beinhaltet, zu entwickeln weil:

- der Anwender nicht gleich zwei verschiedene Befehlssätze (Programmiersprachen) erlernen muß
- die fertigen Programme auf dem Entwicklungssystem, vor Erstellen der Hardware für die betreffende Anwendung bereits getestet werden können und weil
- beim Aufbau dieses Systems schon spezifische Kenntnisse für den betreffenden uP-Typ erworben wurden.

Grundsätzlich kann man, wenn Geschwindigkeit, optimale Speicherausnutzung und erforderlicher externer Hardware-Aufwand außeracht gelassen werden, mit jedem Mikroprozessor *alles* machen. Für den durchschnittlichen ELEKTOR-Leser dürften Anwendungen mit relativ kleinen Programmen, bei welchen die Speicherausnutzung nicht ins Gewicht fällt, ebenso die Geschwindigkeit, dafür aber ein Minimum an Hardware erwünscht ist, am interessantesten sein. Und gerade dafür ist der SC/MP prädestiniert.

Der Aufbau des Entwicklungssystems ist keine unproduktive „ art pour l'art ", sondern der erste Schritt.
Der zweite ist die Lösung, zunächst einfachster Aufgaben mit *eigenen* Programmen.

Viel Erfolg!
Der Verfasser

2. Arbeiten mit der Elbug

2.1 *Der Hardware-Aufbau* bestehend aus CPU-Platine, Hex-I/O-Platine, Input-Unit (Europa-Platine mit dem Sockel für das 3. PROM) und Kassetten-Interface sollte mit Hilfe der RAM-I/0-Karte und den entsprechenden Programmen aus ELEKTOR getestet sein und einwandfrei funktionieren. Nun wird, durch Umlöten der Drahtbrücken die CPU-Platine und die Input-Platine auf Page Ø (Adr. 0000 bis 0FFF) gelegt. Die RAM-I/0-Karte kann auf Page 1 gelegt werden. Sie ist für die Monitor-Software nicht erforderlich; kann aber als preiswerte „ Peripherie " mit dikatischem Wert oder als Einheit mit je 8 I/0-Leitungen (anstatt Datenschalter und LED) weiterhin Verwendung finden.

Die 3 PROM's werden nun unter strengster *Beachtung der Vorsichtsmaßnahmen für MOS-Bauteile* in die entsprechenden Sockel montiert:

PROM 1 . . . . . IC 3 CPU-Platine
PROM 2 . . . . . IC 2 CPU-Platine
PROM 3 . . . . . IC 14 Input-Unit

Beim Anlegen der Versorgungsspannungen (−12 V jetzt auch erforderlich) erscheint „ .. ELBUG " auf dem Display. Falls nicht, liegt ein Fehler in der Hardware vor: „ JMP 2.1."
Das Display ist funktionell in 4 Felder aufgeteilt:

Digit 0 und 1 (die beiden rechten Anzeigen)	= *Datenfeld*
Digit 2 bis 5 (die vier mittleren Anzeigen)	= *Adressfeld*
Digit 6 und 7 (die beiden linken Anzeigen)	= *Kommandofeld*

.
Dezimalpunkte auf dem Display bedeuten, daß das Programm eine Eingabe über die Tastatur erwartet; die beiden Punkte von „ .. ELBUG " verlangen das Betätigen einer Kommando-Taste.
Diese sind innerhalb des Pgrogramms wie folgt definiert:

FØ	=	RUN
EØ	=	MODIFY
DØ	=	SUBTRAKTION
CØ	=	KASSETTE
BØ	=	BLOCK-TRANSFER
AØ	=	CPU-REGISTER
9Ø	=	DOWN
8Ø	=	UP

Die beiden letzten sind keine echten Kommando-Tasten, sondern „ Suffix "-Tasten, welche *nach* Betätigung einer Kommando-Taste eine bestimmte Bedeutung haben, ähnlich wie die von manchen Taschenrechnern bekannten „ Präfix "-Tasten, die *vor* den eigentlichen Kommando-Tasten zu betätigen sind.

2.2 Modify-Kommando

Wird nun (nach „ .. ELBUG ") die Taste „ MODIFY " gedrückt, so erscheint auf dem Display „ MO .... ". Das Programm erwartet eine Adresseingabe über das hexadezimale Tastenfeld. Nach Eingabe des letzten Adress-Digits erscheint sofort auch der Inhalt dieser Adresse auf dem Datenfeld des Display. Nun hat der Benutzer 4 Möglichkeiten:

- Er will den Inhalt dieser Adresse ändern. Dies geschieht durch Eingabe des gewünschten neuen Bytes auf der Hex-Tastatur. Nach der ersten Hex-Taste erscheint das betreffende Hex-Zeichen auf Display 1, Display O erlischt. Nach der 2. Hex-Taste erscheint das Hex-Zeichen auf Display O, aber nur dann, wenn man in die betreffende Adresse auch tatsächlich etwas hineinschreiben kann. Unbeschaltete Adressbereiche (werden von der CUP als X'FF interpretiert) oder PROM-Adressen melden sich wieder mit ihrem (nicht überschreibbarem) Inhalt.
  Hat man sich vertippt, so kann man beliebig oft ein anderes Byte (kein einzelnes Hex-Digit = 1/2 Byte) auf die gleiche Adresse eintasten.

- Man will den Inhalt der nächsthöheren Adresse auf die Anzeige bekommen: „ UP " Taste. Natürlich auch dann, wenn der Inhalt der vorherigen Adresse nicht mit Hex-Eingaben geändert wurde. Auf diese Weise kann recht komfortabel ein Speicherbereich durchgelesen werden.

- Mit der „ DOWN " - Taste ruft man den Inhalt der nächstniedrigeren Adresse ab.

- Die Modify-Routine kann nur mit NRST verlassen werden: „ .. ELBUG ".

Mit Modify bewegt man sich immer in der gleichen Page. Nach 4 FFF folgt 4 ØØØ. Auf Adr. 5000 kommt man mit NRST, MODIFY und 5 Ø Ø Ø.

2.3. Run-Kommando

Hat man z.B. mit Modify ein Programm in ein RAM-Bereich geladen und will es starten, bedient man sich dieses Kommandos.
Zunächst muß mit NRST wieder „ .. ELBUG " auf das Display gebracht werden. Nach Betätigen der RUN-Taste erscheint „ RU ...." auf dem Display Die Startadresse des Programms (die Adresse des ersten Befehlsbytes) wird nun eingetastet. Zum eigentlichen Starten kann jede beliebige Taste (Hex oder Kommando) benutzt werden. Beim Loslassen wird „ RU " auch auf Digit O—1 des Display geschrieben und das Programm gestartet. „ RU ADR RU " bleibt stehen, sofern das betreffende Programm nicht selbst auf das Display zugreift. Geschieht in dem angesprungenen Programm ein XPPC 3 (programmiert oder durch Interrupt) bevor PTR3 neu geladen ist, so erscheint „ ..ELBUG " auf dem Display; Zeichen dafür, daß das User-Programm verlassen wurde und die CPU auf ein neues Kommando wartet.

2.4. Hexadezimale Subtraktion

Dieses Kommando dient zum Errechnen von Displacement-Werten für eigene Programme, als Assemblier-Hilfe.
Nach Drücken dieser Taste zeigt das Display: „ SH.... " = Subtraktion, hexadezimal. (Nicht „ 5 H....", in „ 7-Segment-Schrift " sehen „ 5 " und „ S " nur gleich aus).

Jetzt verlangen die Dezimalpunkte keine Adresse, sondern den Minuenden. Wenn dieser eingegeben ist, erlöschen Digit 6 und 7 und auf Digit 1 erscheint ein Minus-Zeichen. Es folgt die Eingabe des Subtrahenden. Die Differenz bekommt man durch Betätigen einer beliebigen Taste. Sie ist mit einem „ = " versehen. Hier läuft das Programm in eine tote Schleife:

JMP:

JMP JMP 9OFE

Diese kann nur mit NRST verlassen werden.
Negative Differenzen erhält man, so wie sie gebraucht werden: als Zweier-Komlement.

5. Kassetten-Routinen

Zur Ausgabe des Inhaltes eines Speicherbereichs geht man so vor:

NRST „ .. ELBUG "

KA-Taste „ CA . . . . "

Jezt kann man mit der Modify-Taste die gewünschte Ausgabegeschwindigkeit eingegeben werden, s. Tabelle.

MO-Taste	„ CA . . . . MO "
SPEED (HexKB)	„ CaOOXXMO "

Wenn von dieser Möglichkeit kein Gebrauch gemacht wird, so erfolgt die Ausgabe mit 600 Baud (=Bit/sec). Man kann also ohne „ Modify Speed " fortfahren.

Beliebige Taste (außer MO ,UP, DOWN)	„ Ca ....ad "
Anfangsadr. des Speicherbereichs	„ Caxxxx.. "
Endadr. des Speicherbereichs (Hex)	„Cayyyy.. "

Spätestens jetzt muß der Kassetten-Rekorder angeschlossen, auf Aufnahme eingestellt und gestartet werden. Vor dem nächsten Schritt sollte der Rekorder bereits 1 min laufen, damit ein Stück hohe Frequenz (= log 1 = Stopbit) aufgenommen wird. Dieses verlängerte Stopbit wird beim Laden von Kassette dringend benötigt.

DOWN nach einiger Zeit, wenn der Datenblock fertig ausgegeben ist, erscheint auf dem Display „ .. ELBUG "

Die CPU wartet auf das nächste Kommando.

Bei der Ausgabe wird die Anfangs- und Endadresse auf das Band geschrieben. Außerdem wird nach je 32 Datenbytes eine Checksumme (= Arithmetische Summe dieser 32 Bytes, unter Außerachtlassung des Übertrags, also nur 1 Byte) an die Kassette ausgegeben. Der letzte Datenblock kann kürzer sein als 32 Byte (ist aber trotzdem mit einer Checksumme versehen).

Es ist erlaubt, Datenblocks auszugeben, welche Page-Grenzen überschreiten. Z.B. Anfangadr. = OFFO, Endadr. = 1654.

Laden von der Kassette:

NRST „ .. ELBUG "

KA-Taste „ Ca . . . . "

Spätestens hier muß der Rekorder laufen und in seinem Lautsprecher das o.a. verlängerte Stopbit zu hören sein.

UP-Taste nach einer Weile, wenn der Datenblock (auch hörbar) zu Ende ist, erscheint auf dem Display wieder „ .. ELBUG ".

Wenn irgend etwas nicht gestimmt hat — Band beschädigt, Kassetten Interface nicht in Ordnung usw. — erscheint noch während der Datenblock im Lautsprecher zu hören ist

„ Ca Error " auf dem Display.

Das bedeutet, daß die selbst errechnete Checksumme während der Datenübernahme nicht mit der Checksumme vom Band übereinstimmt. Nach „ Error " läuft das Programm in eine tote Schleife, die nur mit NRST verlassen werden kann. Wenn einige Wiederholungsversuche (der Fehler kann auch bei der Ausgabe an die Kassette geschehen sein) nichts bringen, müssen der Logiktester und der Lötkolben in Aktion treten.

Das auf diese Weise geladene Programm wird genau an den Adressen stehen, von wo es an die Kassette ausgegeben wurde; die Anfangs- und Endadressen stehen ja auf dem Band! Soll ein Datenblock von der Kassette auf andere Adressen geladen werden, wird folgendermaßen vorgegangen:

NRST	„ .. ELBUG "
KA-Taste	„ Ca.... "
Beliebige Taste (außer MO, UP, DOWN)	„ Ca.... ad "
Anfangsadr.	„Caxxxx.. "
Endadr.	„Cayyyy.. "
Rekorder laufen lassen	
UP	

Wenn das Interface total defekt ist und ein ewiges Startbit an SIN anliegt, erscheint kein Error weil Ø+Ø = Ø und die Checksumme ja stimmt.
Liegt ein dauerndes Stopbit an SIN, so kommt die Kassettenroutine erst gar nicht in Gang, obwohl der Rekorder schon längst fertig ist.

Wenn nur ein Teil eines Blocks von der Kassette geladen wird (z.B. auf Kassette sind Daten von 1FØØ bis 1 FFF und der erste Teil dieses Blocks soll auf 1F2Ø bis 1F8Ø geladen werden), so erscheint nach korrekt beendeter Übernahme doch „ Error " weil naturgemäß die letzte Checksumme nicht stimmt.

Bei der bisherigen Arbeitsweise wurde angenommen, daß der Datenblock mit 600 Baud an die Kassette ausgegeben wurde. Ist das nicht der Fall, so muß auch hier über Modify die tatsächliche Geschwindigkeit eingegeben werden:

NRST, KA-Taste, MO-Taste, SPEED, (beliebige Taste, Anfangsadr. Endadr.)
Rekorder laufen lassen und UP-Taste.

2.6 Block-Transfer-Kommando

Während des Austestens von Programmen tritt oft die Situation auf, daß an irgendeiner Stelle noch Befehle eingefügt werden müssen oder daß einige Befehle überflüssig sind. Um nicht noch einmal mühe- und fehlervoll das restliche Programm ab dieser Stelle eintippen zu müssen, bedient man sich dieses Kommandos.

Handhabung:

NRST	„ .. ELBUG "
Transfer-Taste	„ BL .... "
Anfangsadr. des Blocks	„BLxxxx .. "
Endadr. des Blocks	„BLyyyy .. "
Neue Anfangsadr.	„BLzzzz .. "
Beliebige Taste	„ .. ELBUG "

Damit ist der Block umgeladen.
Ein Umladen auf eine andere Page ist möglich, aber nicht von oder auf einen Bereich, der eine Page-Grenze überschneidet. Das wäre auch nicht sinnvoll, da ein Programm an der Page-Grenze nur dann weiterläuft, wenn dort mit einem XPPC dieser Sprung ermöglicht wird. Die neue Anfangadr. kann kleiner oder größer sein als die ursprüngliche; ein aufwärts oder abwärts-Umladen ist demnach möglich.

2.7. CPU-Register-Kommando

Dieses Kommando ist auch eine Debug-Hilfe. Damit kann an einer beliebigen Stelle eines zu testenden Programms der Inhalt der CPU-Register auf das Display gebracht werden. Das betreffende Programm wird in ein RAM-Bereich geladen. Dann:

NRST	„.. ELBUG "	
CPU-Taste	„CP .... "	
Startadr. User-Progr.	„CPxxxx.. "	
Stopadr.(Byte nach dem Befehl der als letzter ausgeführt werden soll)	„CPyyyy.. "	
Taste A	„ CP AA "	AA = (A)
Taste E	„CP EE "	EE = (E)
Taste 5 (5 = S 7-Segm.)	„CP 55 "	= (SR)
Taste 1	„CP 1111 "	= (PTR 1)
Taste 2	„CP 2222 "	= (PTR 2)

Die Schleife zum Abfragen dieser 5 Tasten (A, E, 5, 1, 2) wiederholt sich endlos und kann nur mit NRST verlassen werden. Zu beachten ist, daß das Programm nicht läuft wenn (PTR3) im User-Programm verändert wird und daß nachher in der Stopadresse nicht der vorherige Inhalt sondern X'3F(= XPPC 3) steht. Dieses muß vor dem weiteren Austesten wieder in den ursprünglichen Zustand gebracht werden (mit Modify).

RAM-Organisation des LIFO-Stack

ADRESSE	REGISTER	STATUS
0FDF	AC	ERSTER STATUS
0FDE	E	
0FDD	SR	
0FDC	PTR 1 L	
0FDB	PTR 1 H	
0FDA	PTR 2 L	
0FD9	PTR 2 H	
0FD8	PTR 3 L	
0FD7	PTR 3 H	
0FD6	ROUTAD L	
0FD5	ROUTAD H	
0FD4	AC	ZWEITER STATUS
0FD3	E	
0FD2	SR	
0FD1	PTR 1 L	
0FD0	PTR 1 H	
0FCF	PTR 2 L	
0FCE	PTR 2 H	
0FCD	PTR 3 L	
0FCC	PTR 3 H	
0FCB	ROUTAD L	
0FCA	ROUTAD H	
0FC9	AC	DRITTER STATUS usw.

Format der Daten auf Kassette

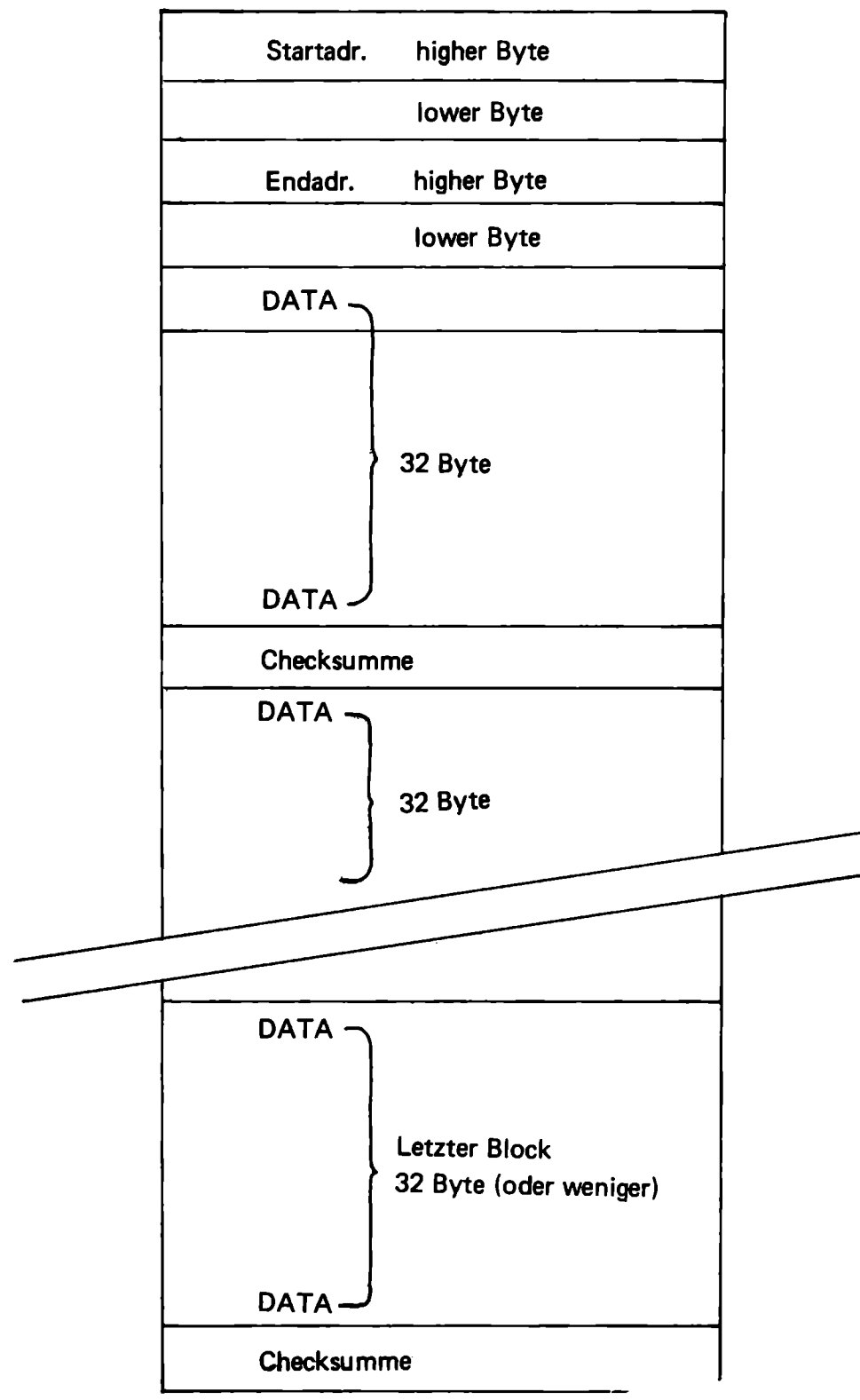


Tabelle mit den Werten zum Laden des Bytes „ SPEED " für verschiedene Geschwindigkeiten der Kassetten-Routinen.

Geschwindigkeit	SPEED	
2400 Bit/sec	ØØØ2	
1200	ØØØ8	
600	ØØ15	(wird vom ELEBUG automatisch geladen)
300 (Kansas City)	ØØ2E	
110 (TTY)	ØØ85	

* Die Werten gelten für SC/MP I mit 1 MHz-Quarz oder SC/MP II mit 2 MHz-Quarz. Für SC/MP II mit 4 MHz-Quarz gelten doppelte Geschwindigkeiten oder die o.a. Werte halbiert (Vorsicht es sind Hex-Zahlen.)

3. Listing

s p e d-8084 inning a.a. leitenstr. 24 zerling tel.(08143)1315

```
                .title          ' e l b u g '
                .page           'stack-routinen'
                .local          /abgrenzung eines '$'-adr.-bereichs
     0700       displ   = 0700  /e.a. des display
     0fff       stakpt  = 0fff  /2 byte f. aktuellen stackpointerstand
     0ffd       routad  = 0ffd  /2 byte adresse der subroutine
     0ffb       stfull  = 0ffb  /'stack-full'-flag
     0ffa       stdeep  = 0ffa  /1 byte f.einstellbare stacktiefe
     0ff9       stkeff  = 0ff9  /aktueller stack-fuellstand
     0ff8       ac      = 0ff8  /zwischenspeicher fuer (ac)
     0ff7       ptr     = 0ff7  /zwischenspeicher fuer (pointer)
     0ff5       speed   = 0ff5  /geschwindigkeit kassetten-routinen

     0fe0       stkbse  = 0fe0  /'stack-base'
     0000       .=0000
                stack:          /programm-anfang
0000 08          nop
0001 c415        ldi x'15       /kassetten-geschwindigkeit auf
0003 c8f1        st speed       /600 bit/sec voreinstellen
0005 c4e0        ldi l(stkbse)  /'ram-stackpointer' auf 'stack-
0007 c8f7        st stakpt      /base' stellen
0009 c40f        ldi h(stkbse)
000b c8f2        st stakpt-1
000d c400        ldi 00
000f c8e9        st stkeff      /stack-zaehler auf 0 stellen
0011 c8e9        st stfull      /stack-full-anzeige = 0
0013 903d        jmp $1         /eigentliches 'elbug' anspringen
                pull:           /status aus stack 'ziehen'
0015 c0e9        ld stakpt      /ptr 1  mit aktuellem stackpointer-
0017 31          xpal 1         /stand laden
0018 c0e5        ld stakpt-1
001a 35          xpah 1
001b c501        ld @1(1)       /routinen-adr. aus stack in
001d c8de        st routad-1    /'routad' laden
001f c501        ld @1(1)
0021 c8db        st routad
0023 c501        ld @1(1)       /ptr 3 aus stack laden
0025 37          xpah 3
0026 c501        ld @1(1)
0028 33          xpal 3
0029 c501        ld @1(1)       /ptr 2 aus stack laden
002b 36          xpah 2
002c c501        ld @1(1)
002e 32          xpal 2
002f c501        ld @1(1)       /(ptr 1) aus stack in zwischen-
0031 c8c4        st ptr-1       /speicher
0033 c501        ld @1(1)
0035 c8c1        st ptr
0037 c501        ld @1(1)       /s-reg. aus stack laden
0039 07          cas
003a c501        ld @1(1)       /e-reg. aus stack laden
003c 01          xae
003d c501        ld @1(1)       /(ac) aus stack in zwischenspeicher
003f c8b8        st ac
```

```
0041 c0b4    ld ptr-1        /aktuellen stackpointerstand in
0043 35      xpah 1          /'stkpt' laden und  ptr 1  aus
0044 c8b9    st stakpt-1     /zwischenspeicher laden
0046 c0b0    ld ptr
0048 31      xpal 1
0049 c8b5    st stakpt
004b b8ad    dld stkeff      /stackfuellstandszeiger up to date
004d c0aa    ld ac           /ac aus zwischenspeicher laden
004f 3f      xppc 3          /ruecksprung in verlassenes programm
0050 9004    jmp push        /notwendig  (s.text)
             $1:
0052 904d    jmp start       /'hilfs-jump'
             $2:
0054 90bf    jmp pull        /dito
             push:           /status in stack ablegen
0056 c8a1    st ac           /(ac) in zwischenspeicher ablegen
0058 c0a6    ld stakpt
005a 33      xpal 3          /(ptr 3) in zwischenspeicher und
005b c89b    st ptr          /ptr 3 als stackpointer laden
005d c0a0    ld stakpt-1
005f 37      xpah 3
0060 c895    st ptr-1
0062 c4ff    ldi l(stakpt)   /(ptr 1) in stack ablegen und
0064 31      xpal 1          /ptr 1  als rampointer laden
0065 cffc    st @-4(3)
0067 c40f    ldi h(stakpt)
0069 35      xpah 1
006a cfff    st @-1(3)
006c 01      xae             /(e) in stack ablegen
006d cb03    st 3(3)
006f 06      csa             /(sr) in stack ablegen
0070 cb02    st 2(3)
0072 c1f9    ld -7(1)        /(ac) aus zwischenspeicher in stack
0074 cb04    st 4(3)
0076 32      xpal 2          /(ptr 2) in stack ablegen
0077 cfff    st @-1(3)
0079 36      xpah 2
007a cfff    st @-1(3)
007c c1f8    ld -8(1)        /(ptr 3) aus zwischenspeicher
007e cfff    st @-1(3)       /holen, in stack ablegen
0080 c1f7    ld -9(1)
0082 cfff    st @-1(3)
0084 c1fe    ld -2(1)        /routinenadr. aus 'routad' in
0086 cfff    st @-1(3)       /stack ablegen
0088 c1fd    ld -3(1)
008a cfff    st @-1(3)       /und  ptr 3 mit routinenadr. laden,
008c 37      xpah 3          /gleichzeitig aktuellen stack-
008d c9ff    st -1(1)        /pointerstand (aus ptr 3)  in
008f c1fe    ld -2(1)        /'stkpt' ablegen.
0091 33      xpal 3
0092 c900    st 0(1)
0094 a9fa    ild -6(1)       /fuellstandszaehler -  up to date
0096 e1fb    xor -5(1)       /mit eingestellter tiefe vergleichen
0098 9c04    jnz $3
009a c4ff    ldi x'ff        /'stack-full'-flag setzen
009c c9fc    st -4(1)
             $3:
009e 3f      xppc 3          /subroutine anspringen
009f 90b3    jmp $2
```

```
                .page              'kommando-schleife'
                .local
                start:
ØØa1 c4ØØ        ldi l(displ)   /ptr 1 zur adressierung des
ØØa3 31          xpal 1         /display und keyboard laden
ØØa4 c4Ø7        ldi h(displ)
ØØa6 35          xpah 1
ØØa7 c4eØ        ldi l(stkbse) /ptr 2 als ram-pointer laden
ØØa9 32          xpal 2
ØØaa c4Øf        ldi h(stkbse)
ØØac 36          xpah 2
ØØad c42f        ldi l(tab1)    /ptr 3  auf tabelle '..elbug'
ØØaf 33          xpal 3
ØØbØ c4Ø1        ldi h(tab1)
ØØb2 37          xpah 3
ØØb3 c4Ø8        ldi Ø8         /zaehler laden
ØØb5 caØb        st Øb(2)
                $1:
ØØb7 c7Ø1        ld e1(3)       / '..elbug' an display
ØØb9 cdØ1        st e1(1)
ØØbb baØb        dld Øb(2)
ØØbd 9cf8        jnz $1         /fertig ?
ØØbf c4Øa        ldi l(ldkb)-1
ØØc1 ca1d        st x' 1d(2)
ØØc3 c4Ø2        ldi h(ldkb)
ØØc5 ca1c        st x' 1c(2)
ØØc7 c4ØØ        js 3(push)     /ldkb, ueber stack, anspringen
ØØc9 37c4
ØØcb 5533
ØØcd 3f
ØØce c48Ø        ldi x' 8Ø      / '.'  an display 3-7
ØØdØ cdfd        st e-3(1)
ØØd2 cdff        st e-1(1)
ØØd4 cdff        st e-1(1)
ØØd6 cdff        st e-1(1)
ØØd8 c4ØØ        ldi ØØ         /display 1 loeschen
ØØda cdff        st e-1(1)
ØØdc c2Ø8        ld 8(2)        /keyboard gesamtes byte, holen
ØØde Ø1          xae
ØØdf 4Ø          lde
ØØeØ e4eØ        xri x' eØ      / = x' eØ ?
ØØe2 9853        jz modify
ØØe4 4Ø          lde
ØØe5 e4fØ        xri x' fØ
ØØe7 9cØ7        jnz $2         / = x' fØ ?
ØØe9 c4Ø1        js 3(run)
ØØeb 37c4
ØØed aØ33
ØØef 3f
                $2:
ØØfØ 4Ø          lde
ØØf1 e4dØ        xri x' dØ      / = x' dØ ?
ØØf3 9cØ7        jnz $3
ØØf5 c4Ø3        js 3(subtr)
ØØf7 37c4
ØØf9 ea33
ØØfb 3f
                $3:
ØØfc 4Ø          lde
```

```
ØØfd e4cØ    xri x'cØ        / = x'cØ ?
ØØff 9cØ7    jnz $4
Ø1Ø1 c4Ø2    js 3(kass)
Ø1Ø3 37c4
Ø1Ø5 f133
Ø1Ø7 3f
             $4:
Ø1Ø8 4Ø       lde
Ø1Ø9 e4bØ    xri x'bØ        / = x'bØ ?
Ø1Øb 9cØ7    jnz $5
Ø1Ød c4Ø5    js 3(transf)
Ø1Øf 37c4
Ø111 4933
Ø113 3f
             $5:
Ø114 4Ø       lde
Ø115 e4aØ    xri x'aØ        / = x'aØ ?
Ø117 9c88    jnz start  ØØA1
Ø119 c4Ø4    js 3(cpu)
Ø11b 37c4
Ø11d 3533                    /keine kommando-taste
Ø11f 3f

             .page            'tabellen'
             tab:            / 7-segm.-code 1-f (Ø=3f=tab-1)
Ø12Ø ØØ      .byte  ØØ,x'5b,x'4f,x'66,x'6d,x'7d,Ø7
Ø121 5b
Ø122 4f
Ø123 66
Ø124 6d
Ø125 7d
Ø126 Ø7
Ø127 7f      .byte x'7f,x'6f,x'77,x'7c,x'58,x'5e,x'79,x'71
Ø128 6f
Ø129 77
Ø12a 7c
Ø12b 58
Ø12c 5e
Ø12d 79
Ø12e 71
             tab1:           / 7-segm.-code '..elbug'
Ø12f ØØ      .byte ØØ,x'3d,x'1c,x'7c,x'38,x'79,x'8Ø,x'8Ø
Ø13Ø 3d
Ø131 1c
Ø132 7c
Ø133 38
Ø134 79
Ø135 8Ø
Ø136 8Ø

             .page            'modify-kommando'
             .local
             modify:
Ø137 c45c     ldi x'5c       / 'mo' an display
Ø139 c9Ø5    st 5(1)
Ø13b c454     ldi x'54
Ø13d c9Ø6    st 6(1)
Ø13f c43e     ldi l(gethex)-1
```

```
Ø141 ca1d    st x'1d(2)
Ø143 3f      xppc 3          /'adr' von keyboard holen
             $1:
Ø144 c2Ø1    ld 1(2)         /ptr 3 auf 'adr' stellen
Ø146 33      xpal 3
Ø147 c2Ø2    ld 2(2)
Ø149 37      xpah 3
Ø14a c3ØØ    ld Ø(3)         / (adr) holen
Ø14c caØØ    st Ø(2)
Ø14e c4aØ    ldi l(puthex)-1
Ø15Ø ca1d    st x'1d(2)
Ø152 c4ØØ    js 3(push)      / adr und (adr) auf display anzeigen
Ø154 37c4
Ø156 5533
Ø158 3f
Ø159 c4Øa    ldi l(ldkb)-1
Ø15b ca1d    st x'1d(2)
Ø15d 3f      xppc 3          /eine taste von kb holen
Ø15e c2Ø1    ld 1(2)
Ø16Ø 33      xpal 3          /ptr 3 wieder auf 'adr'
Ø161 c2Ø2    ld 2(2)
Ø163 37      xpah 3
Ø164 c2Ø8    ld 8(2)         /gesamtes byte resp.taste holen
Ø166 e48Ø    xri x'8Ø        / = 'up'-taste ?
Ø168 98Øa    jz $2
Ø16a e48Ø    xri x'8Ø        /byte wieder herstellen
Ø16c e49Ø    xri x'9Ø        / = 'down'-taste ?
Ø16e 9cØe    jnz $4
Ø17Ø c7ff    ld @-1(3)       /ptr 3 decrementieren
Ø172 9ØØ2    jmp $3
             $2:
Ø174 c7Ø1    ld @1(3)        /ptr 3 incrementieren
             $3:
Ø176 33      xpal 3
Ø177 caØ1    st 1(2)         / (ptr 3) in ram aufbewahren
Ø179 37      xpah 3
Ø17a caØ2    st 2(2)
Ø17c 9Øc6    jmp $1
             $4:
Ø17e c2Ø7    ld 7(2)         / 7-segm.-code fuer  erste taste
Ø18Ø c9ØØ    st Ø(1)         /an display 1 geben
Ø182 c4ØØ    ldi ØØ          /display Ø loeschen
Ø184 c9ff    st -1(1)
Ø186 c2Ø9    ld 9(2)         /binaerwert der taste
Ø188 1e      rr              /in bit 4-7 e
Ø189 1e      rr
Ø18a 1e      rr
Ø18b 1e      rr
Ø18c Ø1      xae
Ø18d c4ØØ    js 3(push)      / 2. hex.-taste abfragen
Ø18f 37c4
Ø191 5533
Ø193 3f
Ø194 c2Ø1    ld 1(2)
Ø196 33      xpal 3          /ptr 3 wieder auf 'adr' stellen
Ø197 c2Ø2    ld 2(2)
Ø199 37      xpah 3
Ø19a c2Ø9    ld 9(2)
Ø19c 58      ore             /byte zusammenstellen und
```

```
019d cb00   st 0(3)          /in 'adr' ablegen
019f 90a3   jmp $1
            .page             'run-kommando'
            .local
            run:
01a1 c450   ldi x'50         /'ru'  an display 6,7
01a3 c906   st 6(1)
01a5 c41c   ldi x'1c
01a7 c905   st 5(1)
01a9 c43e   ldi l(gethex)-1
01ab ca1d   st x'1d(2)       /startadr. von keyboard holen
01ad c400   js 3(push)
01af 37c4
01b1 5533
01b3 3f
01b4 c40a   ldi l(ldkb)-1 /auf beliebige taste warten
01b6 ca1d   st x'1d(2)
01b8 3f     xppc 3
01b9 c201   ld 1(2)
01bb 33     xpal 3           /startadr. in ptr 3
01bc c202   ld 2(2)
01be 37     xpah 3
01bf c7ff   ld @-1(3)        /startadr.-1
01c1 c450   ldi x'50
01c3 c900   st 0(1)          /'ru' an display 0,1
01c5 c41c   ldi x'1c
01c7 c9ff   st -1(1)
01c9 3f     xppc 3           /user-programm anspringen
01ca c40f   js 3(stack)      /bei ruecksprung aus user-programm
01cc 37c4
01ce ff33
01d0 3f
            .local
            .page
            ldbyte:          /routine: 1 byte von kassette holen
01d1 c215   ld x'15(2)
01d3 1c     sr               /speed:2 in ram
01d4 ca14   st x'14(2)
            $1:
01d6 c4ff   ldi x'ff
01d8 01     xae
01d9 19     sio              /stopbit ausgeben
01da 40     lde
01db 9402   jp $2            /auf startbit warten
01dd 90f7   jmp $1
            $2:
01df c4ff   ldi x'ff
01e1 01     xae
01e2 c214   ld x'14(2)       /speed/2  kopieren
01e4 ca0a   st 10(2)
            $3:
01e6 ba0a   dld 10(2)        / 1/2 bit delay
01e8 9cfc   jnz $3
01ea c408   ldi 08           /bit-zaehler laden
01ec ca08   st 8(2)
            $4:
01ee c215   ld x'15(2)       /speed kopieren
01f0 ca09   st 9(2)
01f2 c416   ldi 22           /dly 114 us (sc/mp 1)
```

```
01f4 8f00    dly 00
             $5:
01f6 ba09    dld 9(2)        /speed herunterzaehlen
01f8 9cfc    jnz $5
01fa 19      sio             /bit uebernehmen
01fb ba08    dld 8(2)
01fd 9cef    jnz $4          /8 bit uebernommen?
01ff c215    ld x'15(2)
0201 ca09    st 9(2)
             $6:
0203 ba09    dld 9(2)        /speed herunterzaehlen (1 x = 66 us)
0205 9cfc    jnz $6          / (sc/mp 1)
8207 40      lde             /fertiges byte in ac laden
0208 3f      xppc 3          /ruecksprung
0209 90c6    jmp ldbyte      /fuer erneuten ansprung
             .page            'routine load keyboard'
             .local
             ldkb:
020b c414    ldi l(pull)-1 /ptr 3 vorbereiten
020d 33      xpal 3
020e c400    ldi h(pull)
0210 37      xpah 3
             ldkb1:          /label zum ansprung ohne stack
0211 c401    ldi l(displ)+1
0213 31      xpal 1
0214 c407    ldi h(displ)  /ptr 1 und ptr 2 vorbereiten
0216 35      xpah 1
0217 c4e0    ldi l(stkbse)
0219 32      xpal 2
021a c40f    ldi h(stkbse)
021c 36      xpah 2
             $1:
021d c108    ld 8(1)
021f 94fc    jp $1           /auf tastendruck warten
0221 8f1e    dly 30          /entprellzeit - ca. 30 ms
0223 c108    ld 8(1)
0225 ca08    st 8(2)         /keyboard-ausgang (gesamt) in ram
0227 d40f    ani 0f
0229 ca09    st 9(2)         /binaer-wert der taste in ram und
022b 01      xae             /in e
             $2:
022c c108    ld 8(1)
022e 9402    jp $3           /auf loslassen der taste warten
0230 90fa    jmp $2
             $3:
0232 8f1e    dly 30          /enprellzeit
0234 c41f    ldi l(tab)-1
0236 31      xpal 1
0237 c401    ldi h(tab)
0239 35      xpah 1
023a c180    ld -128(1)      /7-segm.-code holen

023c ca07    st 7(2)         /in ram ablegen
023e 3f      xppc 3          /ruecksprung

             .page            'routine gethex'
             .local
             gethex:
023f c406    ldi l(displ)+6
```

```
0241 31      xpal 1          /ptr 1 auf display 6 stellen
0242 c407    ldi h(displ)
0244 35      xpah 1
0245 c4e7    ldi l(stkbse)+7
0247 32      xpal 2          /ptr 2 auf stack-base + 7
0248 c40f    ldi h(stkbse)
024a 36      xpah 2
024b c404    ldi 04          /tastenzaehler laden
024d caf9    st -7(2)
             $1:
024f c455    ldi l(push)-1
0251 33      xpal 3
0252 c400    ldi h(push)
0254 37      xpah 3
0255 c40a    ldi l(ldkb)-1
0257 cba8    st -88(3)
0259 c402    ldi h(ldkb)
025b cba7    st -89(3)
025d 3f      xppc 3          /'ldkb' (ueber stack) anspringen
025e c4e0    ldi l(stkbse)
0260 33      xpal 3          /ptr 3 jetzt ram-pointer
0261 c40f    ldi h(stkbse)
0263 37      xpah 3
0264 c307    ld 7(3)         / 7-segm.-code holen
0266 cdff    st @-1(1)       /auf display 5 (4,3,2) schreiben
0268 c400    ldi 00
026a c9ff    st -1(1)        /alle display-stellen unterhalb
026c c9fe    st -2(1)        /loeschen
026e c9fd    st -3(1)
0270 c9fc    st -4(1)
0272 c9fb    st -5(1)
0274 c309    ld 9(3)         /binaer-wert der taste in ram-tabelle
0276 ceff    st @-1(2)
0278 bb00    dld 0(3)
027a 9cd3    jnz $1          /4 tasten?
027c c480    ldi x'80
027e c9ff    st -1(1)        / '.'  an display 0,1
0280 c9fe    st -2(1)
0282 c306    ld 6(3)         /higher byte zusammenstellen
0284 1e      rr
0285 1e      rr
0286 1e      rr
0287 1e      rr
0288 01      xae
0289 c305    ld 5(3)
028b 58      ore
028c cb02    st 2(3)
028e c304    ld 4(3)         /lower byte zusammenstellen
0290 1e      rr
0291 1e      rr
0292 1e      rr
0293 1e      rr
0294 01      xae
0295 c303    ld 3(3)
0297 58      ore
0298 cb01    st 1(3)
             jspull:         /hilfslabel zum 'trittbrettfahren'
029a c400    js 3(pull)      /ueber 'pull' zurueck ins hauptprogra
029c 37c4
```

```
Ø29e 1433
Ø2aØ 3f

                .page              'routine puthex'
                .local
                puthex:
Ø2a1 c4eØ        ldi l(stkbse)
Ø2a3 33          xpal 3           /ptr 3 als fixen ram-pointer laden
Ø2a4 c4Øf        ldi h(stkbse)
Ø2a6 37          xpah 3
Ø2a7 c4eØ        ldi l(stkbse)
Ø2a9 32          xpal 2           /ptr 2 u. 1 fuer autoindexierte
Ø2aa c4Øf        ldi h(stkbse)    /adressierung vorbereiten
Ø2ac 36          xpah 2
Ø2ad c4e3        ldi l(stkbse)+3
Ø2af 31          xpal 1
Ø2bØ c4Øf        ldi h(stkbse)
Ø2b2 35          xpah 1
Ø2b3 c4Ø3        ldi Ø3           /byte-zaehler laden
Ø2b5 cbØf        st Øf(3)
                $1:
Ø2b7 c2ØØ        ld Ø(2)          /erstes (naechstes) byte holen
Ø2b9 d4Øf        ani Øf           /bit Ø-3 alleine
Ø2bb cdØ1        st e1(1)         /in ram ablegen
Ø2bd c6Ø1        ld e1(2)         /gleiches byte wieder holen
Ø2bf 1c          sr
Ø2cØ 1c          sr               /bit 4-7
Ø2c1 1c          sr
Ø2c2 1c          sr
Ø2c3 cdØ1        st e1(1)         /in naechste ram-adresse
Ø2c5 bbØf        dld Øf(3)
Ø2c7 9cee        jnz $1           / 3 byte zerlegt?
Ø2c9 c41f        ldi l(tab)-1
Ø2cb 31          xpal 1           /ptr 1 fuer indirekte adressierung
Ø2cc c4Ø1        ldi h(tab)       /vorbereiten
Ø2ce 35          xpah 1
Ø2cf c4Ø6        ldi Ø6           /hex-zeichen-zaehler laden
Ø2d1 cbØf        st Øf(3)
                $2:
Ø2d3 c6Ø1        ld e1(2)         /erstes (naechstes) halbe byte holen
Ø2d5 Ø1          xae
Ø2d6 c18Ø        ld -128(1)       / 7-segm.-code holen

Ø2d8 caØ5        st 5(2)          /in ram zwischenspeichern
Ø2da bbØf        dld Øf(3)
Ø2dc 9cf5        jnz $2           / 6 stellen fertig?
Ø2de c4ØØ        ldi l(displ)
Ø2eØ 31          xpal 1           /ptr 1 auf display stellen
Ø2e1 c4Ø7        ldi h(displ)
Ø2e3 35          xpah 1
Ø2e4 c4Ø6        ldi Ø6           /zaehler laden
Ø2e6 cbØf        st Øf(3)
                $3:
Ø2e8 c6Ø1        ld e1(2)         / 7-segm.-code
Ø2ea cdØ1        st e1(1)         /an display Ø-5
Ø2ec bbØf        dld Øf(3)
Ø2ee 9cf8        jnz $3           /fertig?
Ø2fØ 9Øa8        jmp jspull       /ueber hilfslabel nach pull
```

```
                .page          'kassetten-routinen'
                .local
                kass:
02f2 c439        ldi x'39       /'ca' an display 6,7
02f4 c906        st 6(1)
02f6 c45f        ldi x'5f
02f8 c905        st 5(1)
02fa 01          xae
02fb 19          sio            /stopbit ausgeben
02fc c4ff        ldi x'ff
02fe ca00        st 0(2)        /flag 'adr. von kassette' setzen
0300 c400        js 3(push)     /auf taste warten
0302 37c4
0304 5533
0306 3f
0307 c45f        ldi x'5f       / 'ad' an display 0,1
0309 c900        st 0(1)
030b c45e        ldi x'5e
030d c9ff        st -1(1)
030f c208        ld 8(2)
0311 e4e0        xri x'e0       /taste 'modify' ?
0313 9c1e        jnz $1
0315 c454        ldi x'54       / 'mo' an display 0,1
0317 c900        st 0(1)
0319 c45c        ldi x'5c
031b c9ff        st -1(1)
031d c43e        ldi l(gethex)-1
031f ca1d        st x'1d(2)
0321 3f          xppc 3         / 'speed' von keyboard holen
0322 c201        ld 1(2)
0324 ca15        st x'15(2)
0326 c40a        ldi l(ldkb)-1
0328 ca1d        st x'1d(2)
032a 3f          xppc 3         /auf taste warten
032b c45f        ldi x'5f
032d c900        st 0(1)
032f c45e        ldi x'5e       / 'ad' an display 0,1
0331 c9ff        st -1(1)
                $1:
0333 c208        ld 8(2)
0335 e480        xri x'80       /taste 'up' ?
0337 982c        jz kassup
0339 c43e        ldi l(gethex)-1
033b ca1d        st x'1d(2)
033d 3f          xppc 3         /anfangsadresse von kb holen
033e c201        ld 1(2)
0340 ca0b        st 0b(2)
0342 c202        ld 2(2)
0344 ca0c        st 0c(2)
0346 3f          xppc 3         /end-adrese holen
0347 c40a        ldi l(ldkb)-1
0349 ca1d        st x'1d(2)
034b 3f          xppc 3         /auf taste warten
034c c208        ld 8(2)
034e e480        xri x'80       /taste 'up' ?
0350 9c04        jnz $2
0352 ca00        st 0(2)        /flag 'adressen von kb' setzen
0354 900f        jmp kassup
                $2:
```

```
0356 e480    xri x'80
0358 e490    xri x'90          /taste 'down'  ?
035a 9802    jz $3
035c 9050    jmp jsstak        /wenn andere taste: '..elbug'
            $3:
035e c404    js 3(kassdo)
0360 37c4
0362 e333
0364 3f

            kassup:            /routine: laden von kassette in ram
             local
0365 c41c    ldi x'1c          /'up' an display 0,1
0367 c900    st 0(1)
0369 c473    ldi x'73
036b c9ff    st -1(1)
036d c4d0    ldi l(ldbyte)-1
036f 33      xpal 3            /ptr 3 vorbereiten
0370 c401    ldi h(ldbyte)
0372 37      xpah 3
0373 c200    ld 0(2)
0375 980e    jz $1             /lade-adresse von kassette?
0377 3f      xppc 3            /ladeadresse, higher byte,
0378 ca0c    st 0c(2)
037a 3f      xppc 3            /lower byte,
037b ca0b    st 0b(2)
037d 3f      xppc 3            /stop-adresse, higher byte,
037e ca02    st 2(2)
0380 3f      xppc 3            /lower byte von kassette holen
0381 ca01    st 1(2)
0383 9004    jmp $2
            $1:
0385 3f      xppc 3            / 4 byte von kassette ignorieren
0386 3f      xppc 3
0387 3f      xppc 3
0388 3f      xppc 3
            $2:
0389 c420    ldi 32            /block-laengenzaehler laden
038b ca05    st 5(2)
038d c400    ldi 00
038f ca06    st 6(2)           /ram-byte fuer checksumme loeschen
0391 02      ccl
            $3:
0392 c20b    ld 0b(2)          /ptr 1 auf erste (naechste) lade-
0394 31      xpal 1            /adresse stellen
0395 c20c    ld 0c(2)
0397 35      xpah 1
0398 3f      xppc 3            /daten-byte von kassette holen und
0399 c900    st 0(1)           /in lade-adresse ablegen
039b f206    add 6(2)          /check-summe bilden
039d ca06    st 6(2)
039f 35      xpah 1
03a0 e202    xor 2(2)
03a2 9c11    jnz $4            /stopadresse erreicht?
03a4 31      xpal 1
03a5 e201    xor 1(2)
03a7 9c0c    jnz $4
03a9 3f      xppc 3            /checksumme von kassette holen
03aa e206    xor 6(2)          /mit eigener checksumme identisch?
```

```
03ac 9c21   jnz error
            jsstak:          /hilfs-label (befehls-byte sparen)
03ae c40f    js 3(stack)     /ruecksprung '..elbug'
03b0 37c4
03b2 ff33
03b4 3f
            $4:
03b5 06      csa             / cy/l (von checksumme) in e retten
03b6 01      xae
03b7 02      ccl
03b8 c20b    ld 0b(2)
03ba f401    adi 01          /ptr 1(ueber page-grenze hinweg),
03bc ca0b    st 0b(2)        /incrementieren
03be c20c    ld 0c(2)
03c0 f400    adi 00          /wegen uebertrag (cy/l)
03c2 ca0c    st 0c(2)
03c4 40      lde             / cy/l von checksumme zurueck in sr
03c5 07      cas
03c6 ba05    dld 5(2)        /block-laenge erreicht?
03c8 9cc8    jnz $3
03ca 3f      xppc 3          /checksumme von kassette holen
03cb e206    xor 6(2)        /uebereinstimmung?
03cd 98ba    jz $2
            error:           /'error' an display
03cf c401    ldi l(displ)+1
03d1 31      xpal 1
03d2 c407    ldi h(displ)
03d4 35      xpah 1
03d5 c400    ldi 00
03d7 c904    st 4(1)
03d9 c479    ldi x'79
03db c903    st 3(1)
03dd c450    ldi x'50
03df c902    st 2(1)
03e1 c901    st 1(1)
03e3 c9ff    st -1(1)
03e5 c45c    ldi x'5c
03e7 c900    st 0(1)
            $5:
03e9 90fe    jmp $5          /tote schleife
            .page            'subtraktion'
            .local
            subtr:
03eb c46d    ldi x'6d        / 'sh' an display 6,7
03ed c906    st 6(1)
03ef c476    ldi x'76
03f1 c905    st 5(1)
03f3 c43e    ldi l(gethex)-1
03f5 ca1d    st x'1d(2)
03f7 c400    js 3(push)      /minuend holen
03f9 37c4
03fb 5533
03fd 3f
03fe c440    ldi x'40        / '-' an display 1
0400 c900    st 0(1)
0402 c400    ldi 00
0404 c9ff    st -1(1)
0406 c906    st 6(1)
0408 c905    st 5(1)
```

```
040a c202    ld 2(2)          /minuend in ram ablegen
040c ca14    st x'14(2)
040e c201    ld 1(2)
0410 ca13    st x'13(2)
0412 3f      xppc 3           /subtrahend holen
0413 03      scl              /wegen 2-er komplement
0414 c213    ld x'13(2)
0416 fa01    cad 1(2)         /subtraktion
0418 ca01    st 1(2)
041a c214    ld x'14(2)
041c fa02    cad 2(2)
041e ca02    st 2(2)
0420 c40a    ldi l(ldkb)-1
0422 ca1d    st x'1d(2)
0424 3f      xppc 3           /auf beliebige taste warten
0425 c4a0    ldi l(puthex)-1
0427 ca1d    st x'1d(2)
0429 3f      xppc 3           /differenz an display
042a c400    ldi 00
042c c9ff    st -1(1)
042e c900    st 0(1)
0430 c448    ldi x'48         /'='  an display 6
0432 c905    st 5(1)
             $1:
0434 90fe    jmp $1           /tote schleife
             .page            'cpu-register'
             .local
             cpu:
0436 c439    ldi x'39
0438 c906    st 6(1)          /'cp' an display
043a c473    ldi x'73
043c c905    st 5(1)
043e c43e    ldi l(gethex)-1
0440 ca1d    st x'1d(2)
0442 c400    js 3(push)       /startadresse user-programm holen
0444 37c4
0446 5533
0448 3f
0449 c201    ld 1(2)
044b ca0e    st 0e(2)         /startadresse umladen
044d c202    ld 2(2)
044f ca0d    st 0d(2)
0451 3f      xppc 3           /stopadresse holen
0452 c201    ld 1(2)
0454 31      xpal 1           /ptr 1  auf stopadresse
0455 c202    ld 2(2)
0457 35      xpah 1
0458 c43f    ldi x'3f         /xppc 3 (=x'3f) in stopadresse
045a c900    st 0(1)
045c c471    ldi l($1)-1      /'$1'  in routad laden
045e ca1d    st x'1d(2)
0460 c404    ldi h($1)
0462 ca1c    st x'1c(2)
0464 c20e    ld 0e(2)         /ptr 2 zum ansprung des user-
0466 01      xae              /programms vorbereiten
0467 c20d    ld 0d(2)
0469 36      xpah 2
046a 40      lde
046b 32      xpal 2
```

```
046c c6ff    ld e-1(2)
046e c455    ldi l(push)-1 /ptr 3  fuer den ruecksprung
0470 33      xpal 3        /nach 'push' vorbereiten
0471 3e      xppc 2        /user-programm-ansprung
             $1:
0472 c4e0    ldi l(stkbse) /ptr 2 wieder als ram-pointer laden
0474 32      xpal 2
0475 c40f    ldi h(stkbse)
0477 36      xpah 2
0478 c4d5    ldi l(stkbse)-11
047a ca1f    st x'1f(2)    / 'stkpt' um eine etage verfaelschen
047c c40a    ldi l(ldkb)-1
047e ca1d    st x'1d(2)
0480 c402    ldi h(ldkb)
0482 ca1c    st x'1c(2)
0484 c400    js 3(push)    /tastendruck abwarten
0486 37c4
0488 5533
048a 3f
048b c208    ld 8(2)       /keyboard-byte in e
048d 01      xae
048e c4a0    ldi l(puthex)-1
0490 ca1d    st x'1d(2)    /'routad' mit puthex-1 laden
0492 c401    ldi l(displ)+1
0494 31      xpal 1        /ptr 1 wieder auf display
0495 c407    ldi h(displ)
0497 35      xpah 1
0498 40      lde
0499 e4fa    xri x'fa      /taste 'a' ?
049b 9816    jz $2
049d 40      lde
049e e4fe    xri x'fe      /taste 'e' ?
04a0 9815    jz $3
04a2 40      lde
04a3 e4f5    xri x'f5      /taste '5' ?  (fuer sr)
04a5 9814    jz $4
04a7 40      lde
04a8 e4f1    xri x'f1      /taste '1' ?
04aa 9813    jz $5
04ac 40      lde
04ad e4f2    xri x'f2      /taste '2' ?
04af 9815    jz $6
             $11:
04b1 90bf    jmp $1
             $2:
04b3 c2ff    ld -1(2)      / (ac) aus stack holen
04b5 901c    jmp $8
             $3:
04b7 c2fe    ld -2(2)      / (e) aus stack holen
04b9 9018    jmp $8
             $4:
04bb c2fd    ld -3(2)      / (sr) aus stack holen
04bd 9014    jmp $8
             $5:
04bf c2fc    ld -4(2)
04c1 01      xae
04c2 c2fb    ld -5(2)      / (ptr 1) aus stack holen
04c4 9005    jmp $7
             $6:
```

```
04c6 c2fa    ld -6(2)        / (ptr 2) aus stack holen
04c8 01      xae
04c9 c2f9    ld -7(2)
             $7:
04cb ca02    st 2(2)         /register-inhalte fuer puthex
04cd 40      lde             /umladen
04ce ca01    st 1(2)
04d0 3f      xppc 3          /puthex-ansprung
04d1 9009    jmp $9
             $8:
04d3 ca01    st 1(2)         /register-inhalte umladen
04d5 3f      xppc 3          /puthex-ansprung
04d6 c400    ldi 00
04d8 c903    st 3(1)         /restliches display loeschen
04da c904    st 4(1)
             $9:
04dc c400    ldi 00
04de c900    st 0(1)
04e0 c9ff    st -1(1)
04e2 90cd    jmp $11         /sprung nach $1 (ueber $11)
             .page          'ausgabe an kassette'
             .local
             kassdo:
04e4 c45e    ldi x'5e
04e6 c900    st 0(1)         / 'do' an display 0,1
04e8 c45c    ldi x'5c
04ea c9ff    st -1(1)
04ec c20b    ld 0b(2)        /ptr 1  auf anfangs-lade-adresse
04ee 31      xpal 1
04ef c20c    ld 0c(2)
04f1 35      xpah 1
04f2 c4d7    ldi l(bytout)-1
04f4 33      xpal 3          /ptr 3 auf byte-ausgaberoutine
04f5 c405    ldi h(bytout)
04f7 37      xpah 3
04f8 c20c    ld 0c(2)        /higher byte anfangs-ladeadresse
04fa 3f      xppc 3          /an kassette
04fb c20b    ld 0b(2)
04fd 3f      xppc 3          /lower byte
04fe c202    ld 2(2)
0500 3f      xppc 3          /higher byte end-adresse
0501 c201    ld 1(2)
0503 3f      xppc 3          /lower byte
             $1:
0504 c420    ldi 32          /block-laengen-zaehler laden
0506 ca05    st 5(2)
0508 c400    ldi 00
050a ca06    st 6(2)         /ram-byte fuer checksumme loeschen
050c 02      ccl
             $2:
050d c100    ld 0(1)         /erstes (naechstes) datenbyte holen
050f 01      xae
0510 c206    ld 6(2)
0512 70      ade             /checksumme bilden
0513 ca06    st 6(2)
0515 40      lde
0516 3f      xppc 3          /datenbyte an kassette ausgeben
0517 35      xpah 1
0518 e202    xor 2(2)        /ptr 1  =  end-adresse ?
```

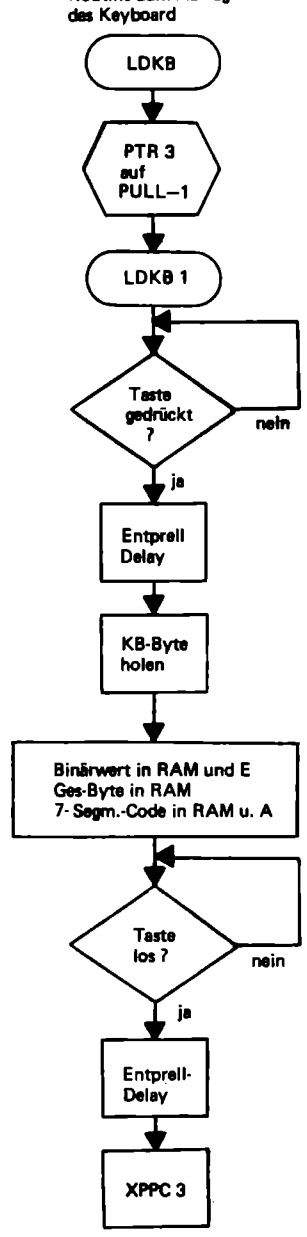
```
051a 01      xae
051b 40      lde
051c e202    xor 2(2)
051e 35      xpah 1
051f 40      lde
0520 9c08    jnz $3
0522 31      xpal 1
0523 e201    xor 1(2)
0525 9819    jz $4
0527 e201    xor 1(2)
0529 31      xpal 1
             $3:
052a 06      csa            / cy/l (von checksumme) in e retten
052b 01      xae
052c 02      ccl
052d 31      xpal 1         /ptr 1 ueber page-grenze
052e f401    adi 01         /hinweg incrementieren
0530 31      xpal 1
0531 35      xpah 1
0532 f400    adi 00
0534 35      xpah 1
0535 40      lde
0536 07      cas            / cy/l zurueck
0537 ba05    dld 5(2)       /block fertig ?
0539 9cd2    jnz $2
053b c206    ld 6(2)
053d 3f      xppc 3         /checksumme an kassette
053e 90c4    jmp $1
             $4:
0540 c206    ld 6(2)
0542 3f      xppc 3         /checksumme an kassette
             jsstac:
0543 c40f    js 3(stack)    /ruecksprung: '..elbug'
0545 37c4
0547 ff33
0549 3f

             .page
             .local          'block-transfer-kommando'
             transf:
054a c47c    ldi x'7c       / 'bl' an display 6,7
054c c906    st 6(1)
054e c438    ldi x'38
0550 c905    st 5(1)
0552 c43e    ldi l(gethex)-1
0554 ca1d    st x'1d(2)
0556 c400    js 3(push)     /block-anfangs-adresse holen (ba)
0558 37c4
055a 5533
055c 3f
055d c201    ld 1(2)
055f ca10    st x'10(2)
0561 c202    ld 2(2)
0563 ca0f    st 0f(2)
0565 3f      xppc 3         /block-ende-adresse holen (be)
0566 c201    ld 1(2)
0568 ca0e    st 0e(2)
056a c202    ld 2(2)
056c ca0d    st 0d(2)
056e 3f      xppc 3         /neue block-anfangs-adresse holen
```

```
 856f c40a   ldi l(ldkb)-1 / = 'nba'
0571 ca1d     st x' 1d(2)
0573 3f      xppc 3          /auf beliebige taste wartenk
0574 03      scl
0575 c201    ld 1(2)         /subtraktion: nba - ba
0577 fa10    cad x' 10(2)
0579 ca0c    st 0c(2)
057b c202    ld 2(2)
057d fa0f    cad 0f(2)
057f ca0b    st 0b(2)
0581 9429    jp $3           /differenz = positiv?
0583 c210    ld x' 10(2)
0585 31      xpal 1          /ptr 1 auf 'ba'
0586 c20f    ld 0f(2)
0588 35      xpah 1
0589 c201    ld 1(2)
058b 33      xpal 3          /ptr 3 auf 'nba'
058c c202    ld 2(2)
058e 37      xpah 3
             $1:
058f c501    ld @1(1)
0591 cf01    st @1(3)        /block umladen
0593 c5ff    ld @-1(1)       /ptr 1 -1
0595 31      xpal 1
0596 e20e    xor 0e(2)       / = be?
0598 01      xae
0599 40      lde
059a e20e    xor 0e(2)       / (ptr 1 l) wieder herstellen
059c 31      xpal 1
059d 40      lde
059e 9c08    jnz $2
05a0 35      xpah 1
05a1 e20d    xor 0d(2)
             $11:            /hilfslabel
05a3 989e    jz jsstac       /ruecksprung '..elbug'
05a5 e20d    xor 0d(2)       / (ptr 1 h) wieder herstellen
05a7 35      xpah 1
             $2:
05a8 c501    ld @1(1)        /ptr 1 +1
05aa 90e3    jmp $1
             $3:
05ac c20e    ld 0e(2)        /ptr 1 auf be
05ae 31      xpal 1
05af c20d    ld 0d(2)
05b1 35      xpah 1
05b2 c501    ld @1(1)        /ptr 1 +1
05b4 03      scl
05b5 c20e    ld 0e(2)        /ptr 3 auf  be + nba - ba + 1
05b7 f20c    add 0c(2)
05b9 33      xpal 3
05ba c20d    ld 0d(2)
05bc f20b    add 0b(2)
05be 37      xpah 3
             $4:
05bf c5ff    ld  @-1(1)      /block umladen
05c1 cfff    st @-1(3)
05c3 31      xpal 1
05c4 e210    xor x' 10(2)    /ptr 1 = ba ?
05c6 01      xae
```

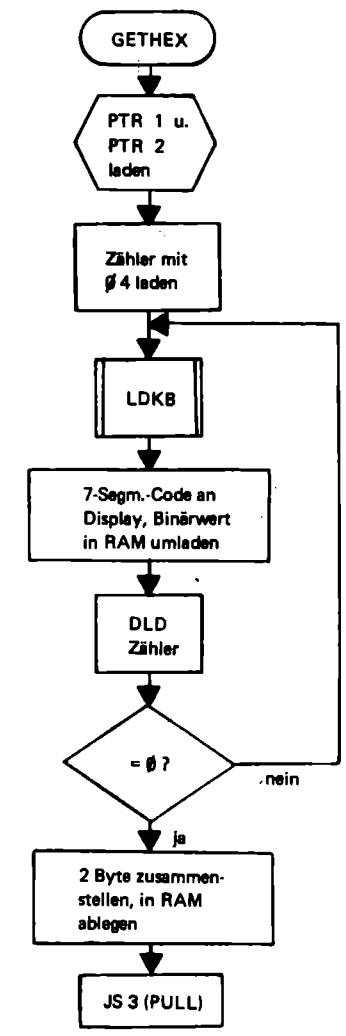
```
Ø5c7 4Ø       lde
Ø5c8 e21Ø     xor x'1Ø(2)
Ø5ca 31       xpal 1
Ø5cb 4Ø       lde
Ø5cc 9cf1     jnz $4
Ø5ce 35       xpah 1
Ø5cf e2Øf     xor Øf(2)
Ø5d1 98dØ     jz $11
Ø5d3 e2Øf     xor Øf(2)
Ø5d5 35       xpah 1
Ø5d6 9Øe7     jmp $4
              .local
              bytout:          / 1 byte an kassette ausgeben
Ø5d8 caØ7     st 7(2)          /byte in ram aufbewahren
Ø5da c4Øb     ldi 11
Ø5dc caØ8     st 8(2)          /bit-zaehler laden
Ø5de c4ØØ     ldi ØØ
Ø5eØ Ø1       xae
Ø5e1 19       sio              /startbit ausgeben
Ø5e2 Ø1       xae              /toter befehl
Ø5e3 ba2Ø     dld x'2Ø(2)      /toter befehl zum zeitausgleich
Ø5e5 c2Ø7     ld 7(2)
Ø5e7 Ø1       xae              /byte in e
              $1:
Ø5e8 c4Øb     ldi 11
Ø5ea 8fØØ     dly ØØ           /dly 7Ø us (sc/mp 1)
Ø5ec c215     ld x'15(2)       /'speed' kopieren
Ø5ee caØ9     st 9(2)
              $2:
Ø5fØ baØ9     dld 9(2)         /speed decrementieren
Ø5f2 9cfc     jnz $2
Ø5f4 19       sio              /bitausgabe
Ø5f5 4Ø       lde
Ø5f6 dc8Ø     ori x'8Ø         /stopbit an byte an fuegen
Ø5f8 Ø1       xae
Ø5f9 baØ8     dld 8(2)
Ø5fb 9ceb     jnz $1           /wenn bitzaehler = Ø ,weiter
Ø5fd 3f       xppc 3           /ruecksprung
Ø5fe 9Ød8     jmp bytout       /fuer erneuten ansprung
              .end
```

4. Flußdiagramme

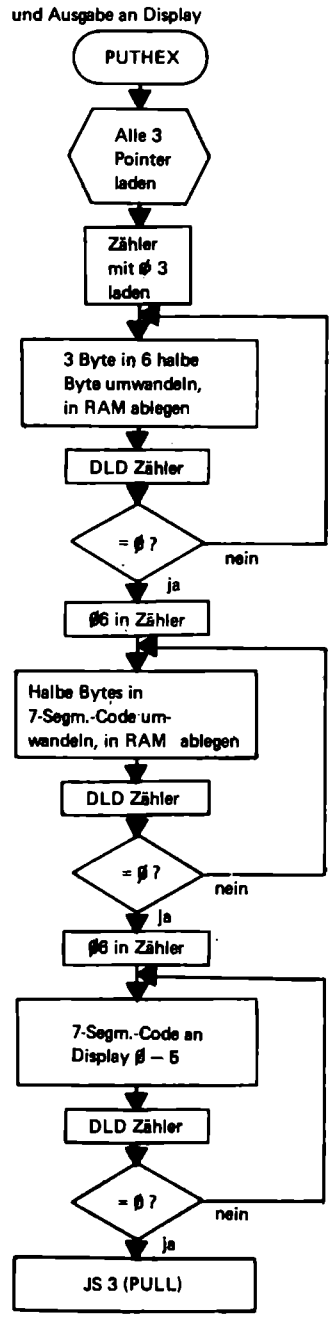
Routine zum Abfragen des Keyboard



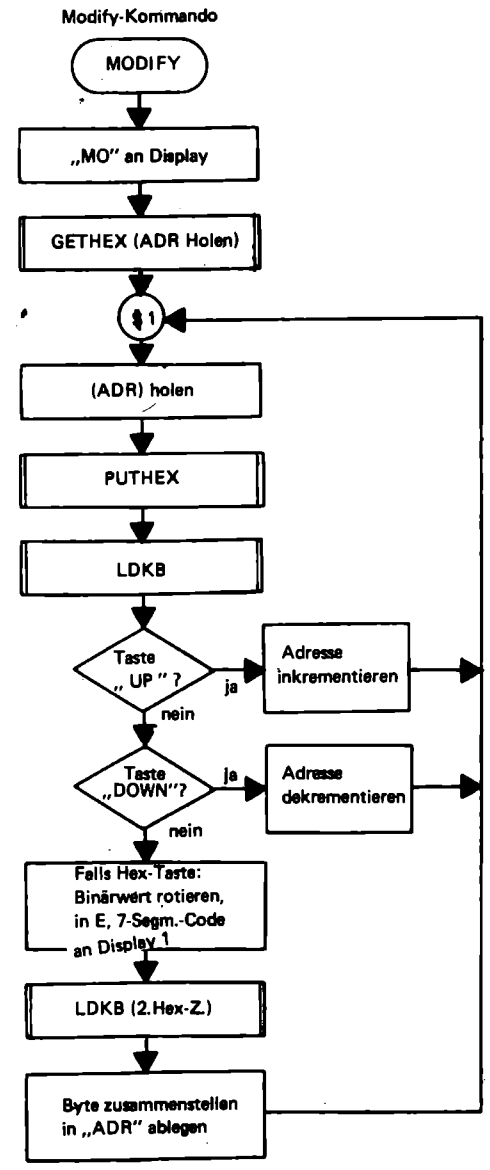
Routine zum Umwandeln von 4 Hex-Zeichen in 2 binäre Byte



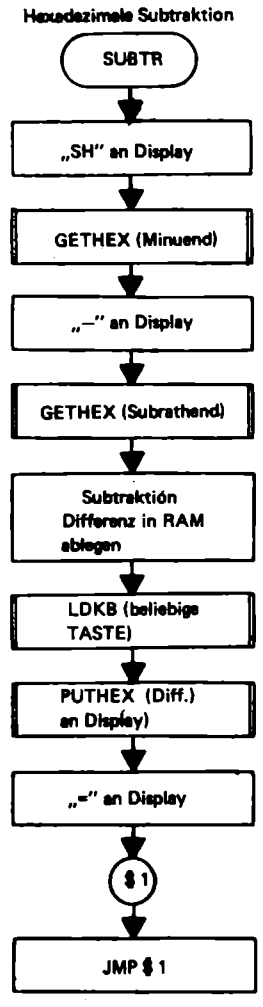
Routine zum Umwandeln von 3 binären Bytes in 6 Hex-Zeichen und Ausgabe an Display



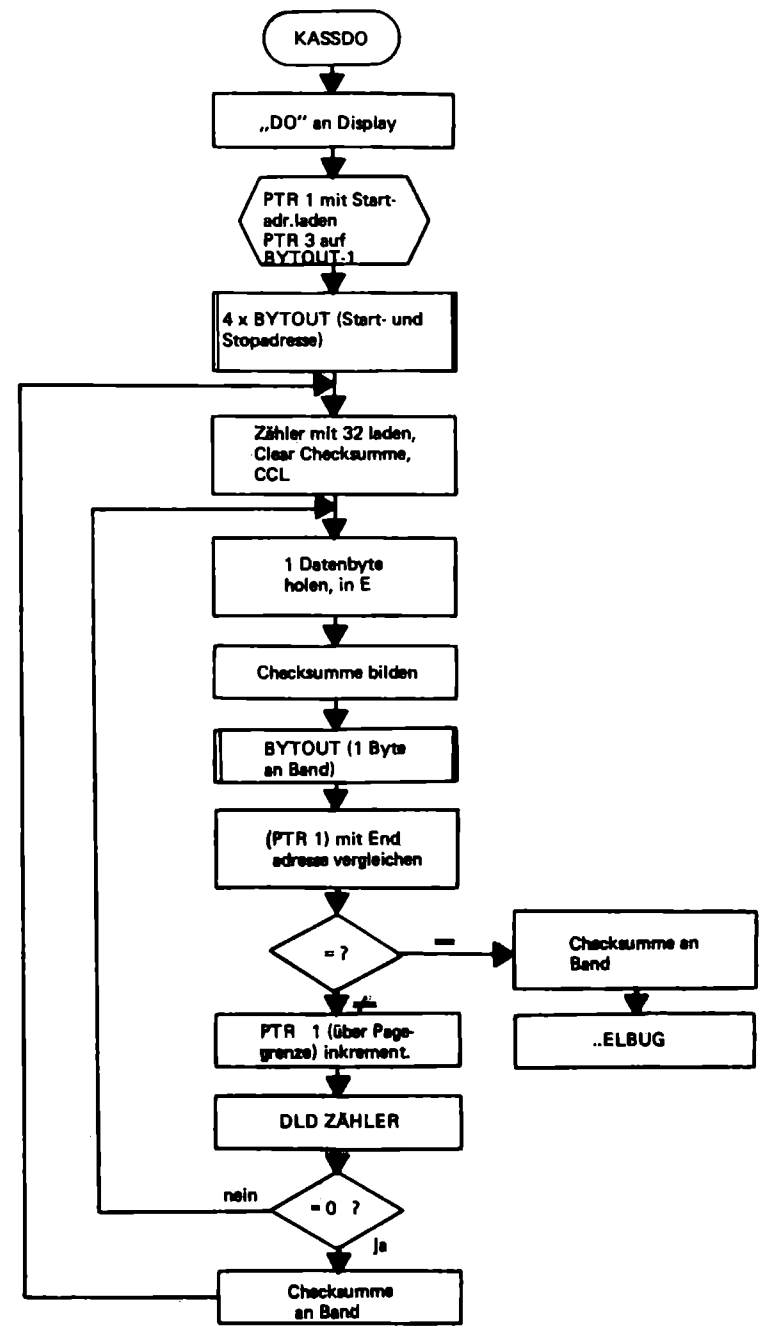
Modify-Kommando



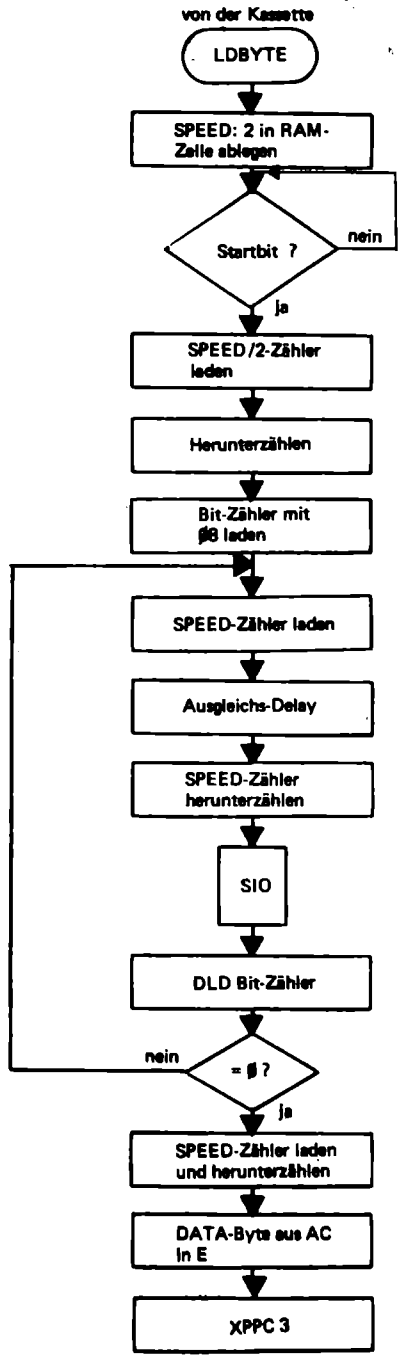
Hexadezimale Subtraktion



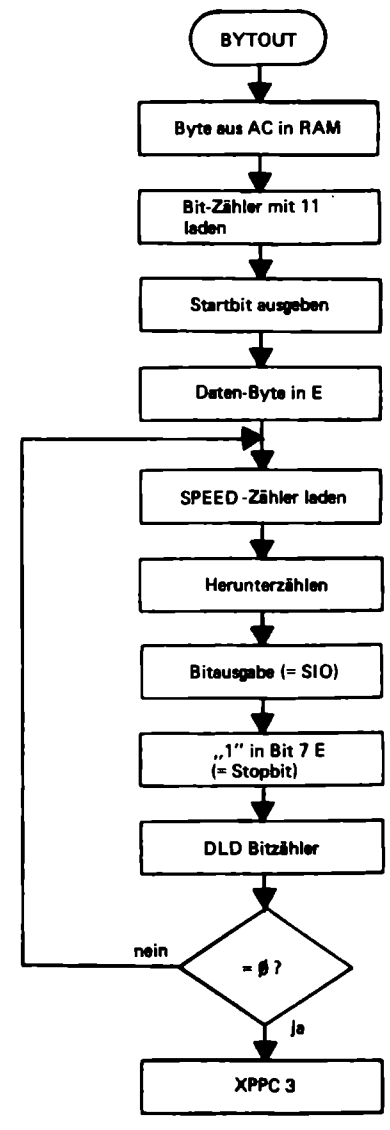
Kassetten-Routine zur Ausgabe von Daten an die Kassette

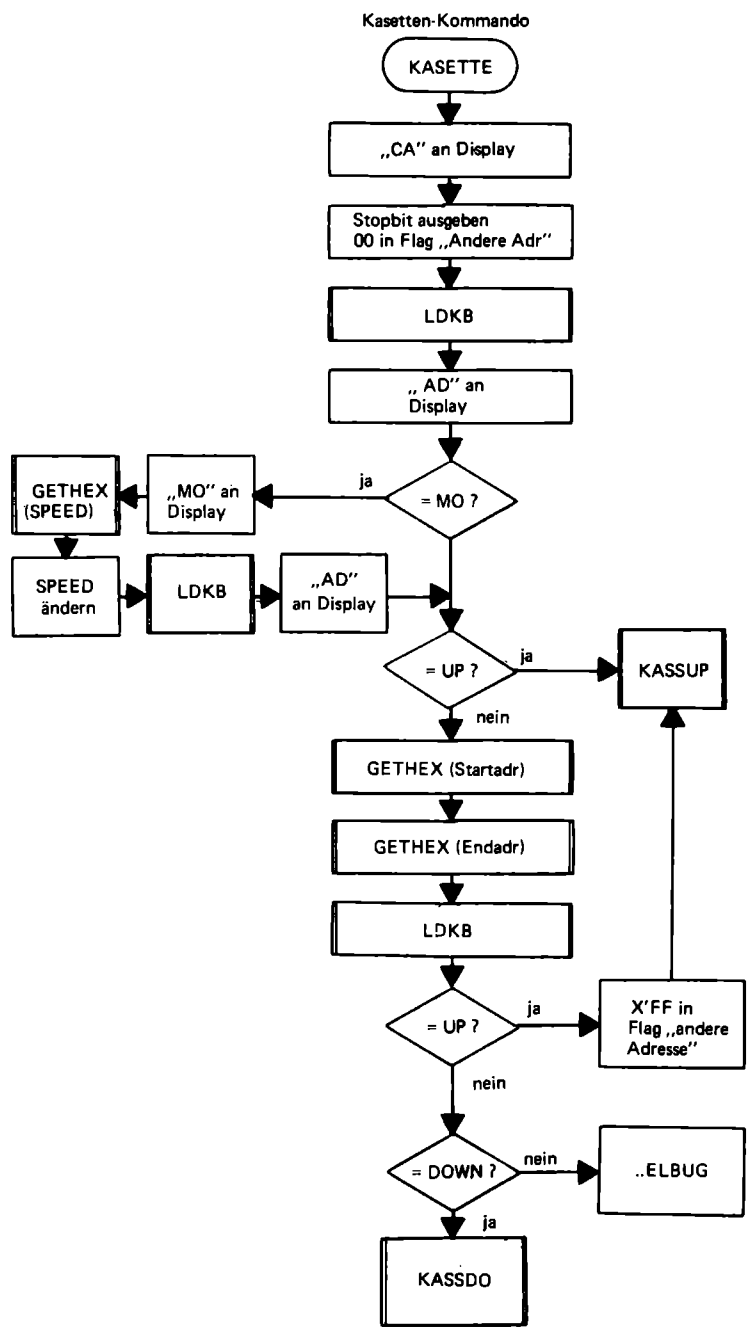


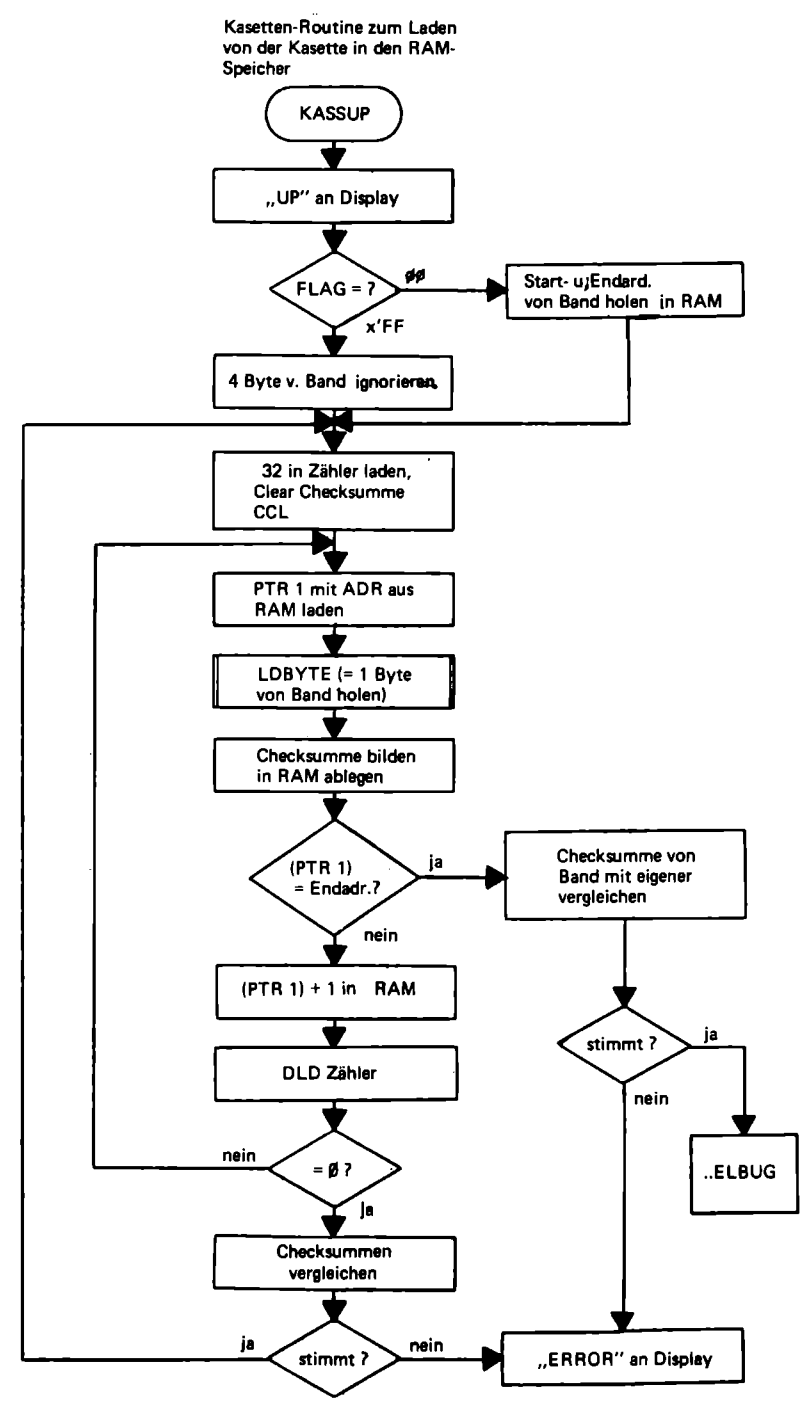
Routine zum Holen eines Byte von der Kassette



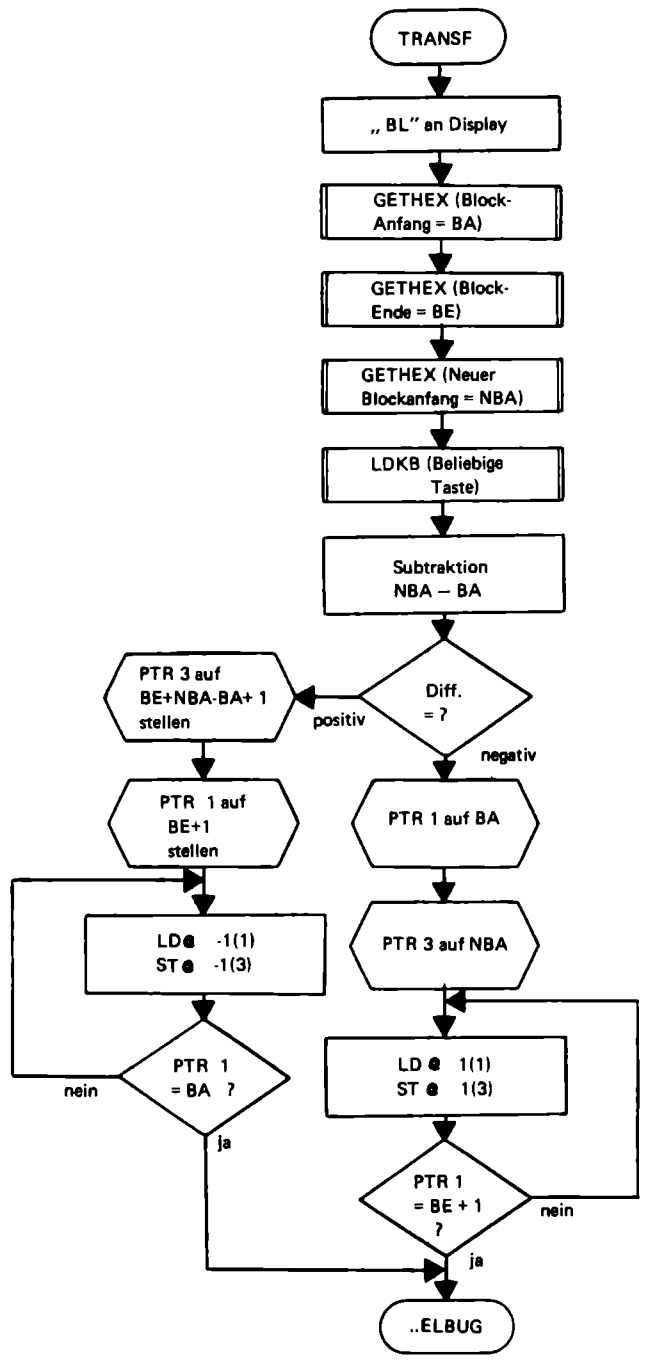
Routine zur Ausgabe eines Byte an die Kassette



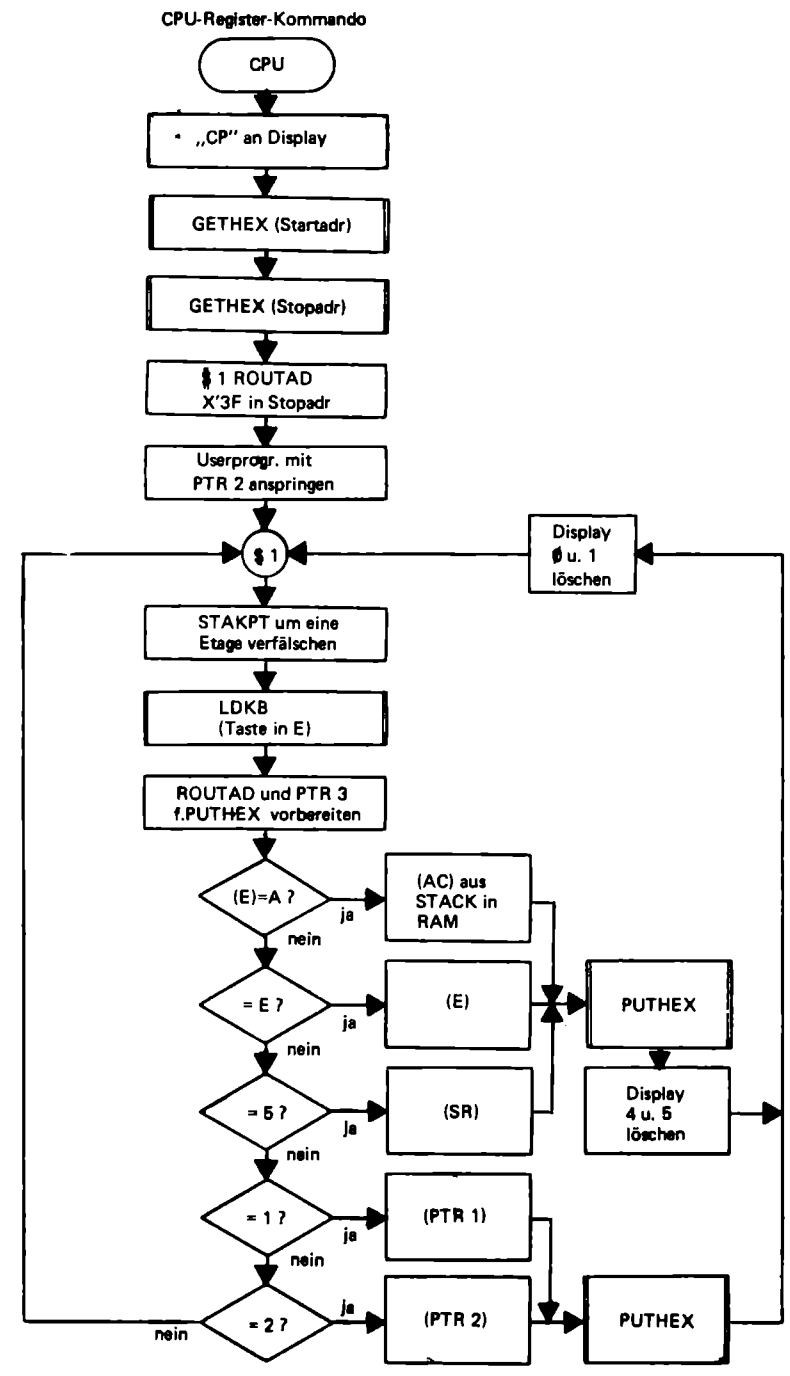

Kasetten-Kommando
KASETTE
„CA" an Display
Stopbit ausgeben
00 in Flag „Andere Adr"
LDKB
„ AD" an
Display
= MO ?
ja
„MO" an
Display
GETHEX
(SPEED)
SPEED
ändern
LDKB
„AD"
an Display
= UP ?
ja
KASSUP
nein
GETHEX (Startadr)
GETHEX (Endadr)
LDKB
= UP ?
ja
X'FF in
Flag „andere
Adresse"
nein
= DOWN ?
nein
..ELBUG
ja
KASSDO


Kasetten-Routine zum Laden
von der Kasette in den RAM-
Speicher
KASSUP
„UP" an Display
FLAG = ?
ØØ
Start- u.Endard.
von Band holen in RAM
x'FF
4 Byte v. Band ignorieren
32 in Zähler laden,
Clear Checksumme
CCL
PTR 1 mit ADR aus
RAM laden
LDBYTE (= 1 Byte
von Band holen)
Checksumme bilden
in RAM ablegen
(PTR 1)
= Endadr.?
ja
Checksumme von
Band mit eigener
vergleichen
nein
(PTR 1) + 1 in RAM
stimmt ?
ja
..ELBUG
nein
DLD Zähler
nein
= Ø ?
ja
Checksummen
vergleichen
ja
stimmt ?
nein
„ERROR" an Display

Transfer von Datenblöcken



CPU-Register-Kommando



5. Erläuterungen zu den einzelnen Kommandos und Routinen

Die Reihenfolge der Kommandos und Subroutinen innerhalb des ELBUG scheint zunächst etwas kunterbunt gemischt: dies hat aber einen bestimmten Zweck.

- Innerhalb des ersten K (0000 - 03FF) sind alle Subroutinen und die Kommandos MODIFY RUN und KASSUP (= Laden eines RAM-Bereichs von der Kassette) untergebracht.
  Dadurch ist eine hardwaremäßige Anwender-Konfiguration, bestehend aus Hex-I/O-Platine, CPU-Platine, einer 4k RAM-Platine und Kassetten-Interface möglich. Der Adressdekoder(von der Input-Unit-Platine) müßte noch auf einer kleinen Wire-Wrap-Platine (und eventuelle Interfaces) aufgebaut werden. Mit dieser „ Minimal-Konfiguration " ließe sich z.B. eine Maschinen-Steuerung realisieren. Auf dem „ großen System " wird das notwendige Programm entwickelt getestet, auf die Kassette geschrieben und dann „ vor Ort " in das kleine System geladen und mit RUN gestartet, kleine Änderungen wären mit MODIFY noch möglich.

- Die Stack-Routinen *müssen* sich nahe am Anfang des Programms befinden, weil diese zum Teil die PC-relative Adressierung benötigen. Durch den Page-Mechanismus wird das RAM von Adr. 0F00 damit erreicht, so als ob es sich direkt unterhalb von Adr. 0000 befinden würde.

- Dadurch, daß die Subroutinen LDKB, GETHEX und PUTHEX innerhalb des gleichen 1/4-Bereichs liegen (higher Byte der Adr. identisch = 02) können etliche Befehlsbyte eingespart werden. Näheres s. weiter unten.

- Kommandos, welche mit einem Rücksprung in die Kommandoschleife beendet sind, wurden nahe zusammengelegt, um mit einem kurzen Sprung (PC-relativ) zum langen Sprung (PTR-relativ) des Nachbarn zu gelangen. Dieses „ Trittbrett-Fahren " spart wieder einige Befehlsbyte.
  Die ersten Befehle stellen eine sogenannte „ *Initialize* " Routine dar (bis „ PULL "). Hier werden einige RAM-Byte definiert geladen.
  Anschließend wird über einen Hilfsjump die Kommando-Schleife angesprungen. Die Stack-Routinen werden dabei übersprungen.

Die *Kommando-Schleife* hat folgende Aufgaben:

- PTR 1 und PTR 2 laden
- ..ELBUG auf das Display schreiben (das Programm „ meldet sich")
- die Keyboard-Routine anspringen und je nach gedrückter Taste
- die entsprechende Kommando-Routine anspringen.

Bei Verlassen der Kommandoschleife enthält PTR 1 die EA des Display (Digit 1), PTR 2 ist als RAM-Pointer geladen (STKBSE) und auf dem Display steht „ ..... " (nicht sichtbar, weil unmittelbar danach das betreffende Kommando auf das Kommando-Feld geschrieben wird.

Die *Stack-Routinen* des ELBUG stellen einen Software-LIFO-Stack mit Abspeichern aller CPU-Register-Inhalte, außer PC, dar, der für programmiertes oder asynchrones (INTERRUPT) Anspringen von Subroutinen, sequentiell oder geschachtelt (nested) verwendbar ist. Für die Funktion des ELBUG wäre diese Komplexität nicht erforderlich gewesen, ist aber für Anwender-Programme, besonders bei Interrupt-Verarbeitung sehr nützlich.
Um diese Aufgaben zu erfüllen, ist zunächst ein RAM notwendig. Die LIFO-Funktion wird nicht so erfüllt, daß die aufeinanderfolgend abgespeicherten STATEN im RAM hinunter und hinauf geschoben werden, sondern ein Pointer wird als STACK-POINTER benutzt. Er zeigt jeweils auf die RAM-Adr. des zuletzt abgespeicherten STATUS. Damit ist aber (die nicht sehr üppige) Anzahl der Index-Register des SC/MP bereits auf 2 zusammengeschmolzen.

R A M – T A B E L L E

ADR	STACK	LDKB	GETHEX	PUTHEX	MODIFY	SUBTR	KASS	CPU	TRANSF
0FFF	STAKPT, lower								
0FFE	STAKPT,higher								
0FFD	ROUTAD,lower								
0FFC	ROUTAD,higher								
0FFB	STFULL								
0FFA	STDEEP								
0FF9	STKEFF								
0FF8	AC								
0FF7	PTR,lower								
0FF6	PTR,higher								
0FF5	SPEED						SPEED		
0FF4						Minuend,high.	0,5 SPEED		
0FF3						Minuend,lower			
0FF2									
0FF1									
0FF0									BA,lower
0FEF				Zähler					BA,higher
0FEE								Start,lower	BE,lower
0FED								Start,higher	BE,higher
0FEC				7-Segm-			Start,higher		Diff,lower
0FEB				Code	Zähler		Start,lower		Diff,higher
0FEA					1/2 Byte		0,5 SPEED		
0FE9		Taste,binär					SPEED		
0FE8		Taste,gesamt					Count Bit		
0FE7		7-Segm-Code					DATA		
0FE6				halbe			Checksumme		
0FE5			Tasten	Bytes			Count,Byte		
0FE4			binär						
0FE3									
0FE2			Byte,higher	ADR,higher	ADR,higher	Diff.,higher	Stop.,higher		NBA,higher
0FE1			Byte,lower	ADR,lower	ADR,lower	Diff.,lower	Stop.,lower		NBA,lower
0FE0	STKBSE		Zähler	DATA	DATA		Flag „ADR"		

Abb. 1 Prinzipielle Arbeitsweise des Software-LIFO-Stack

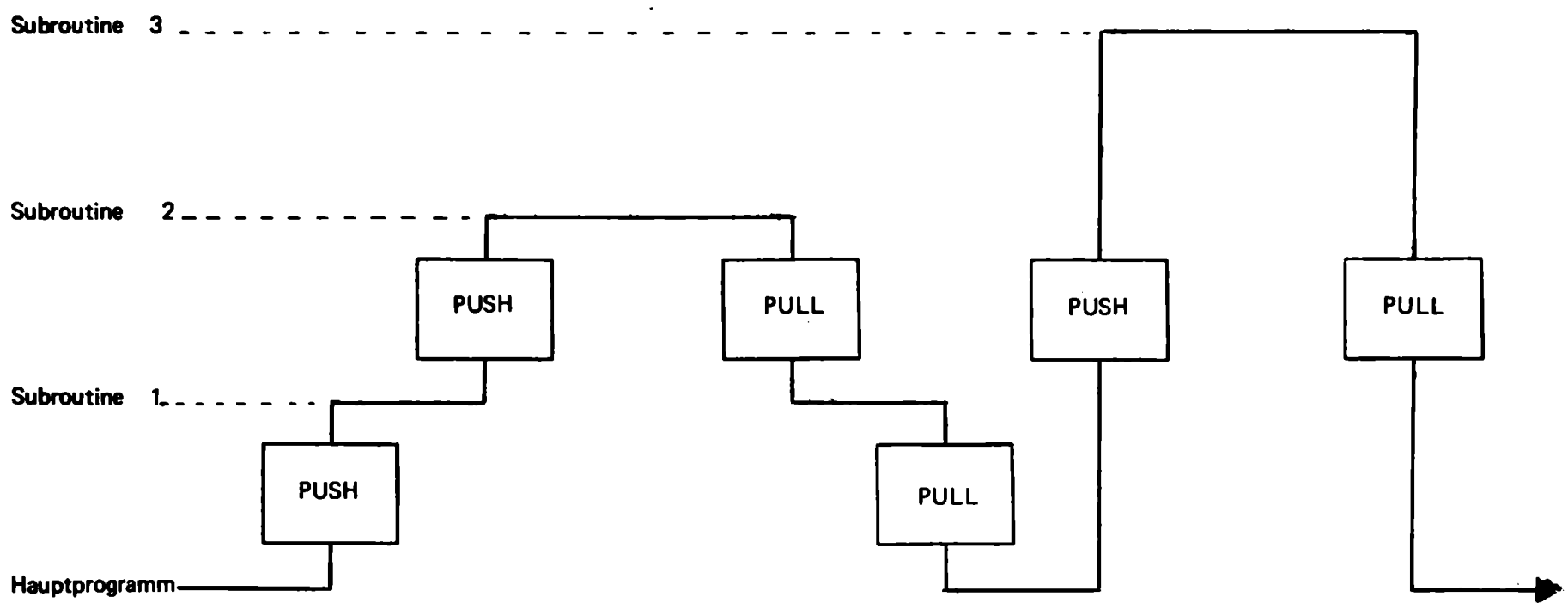
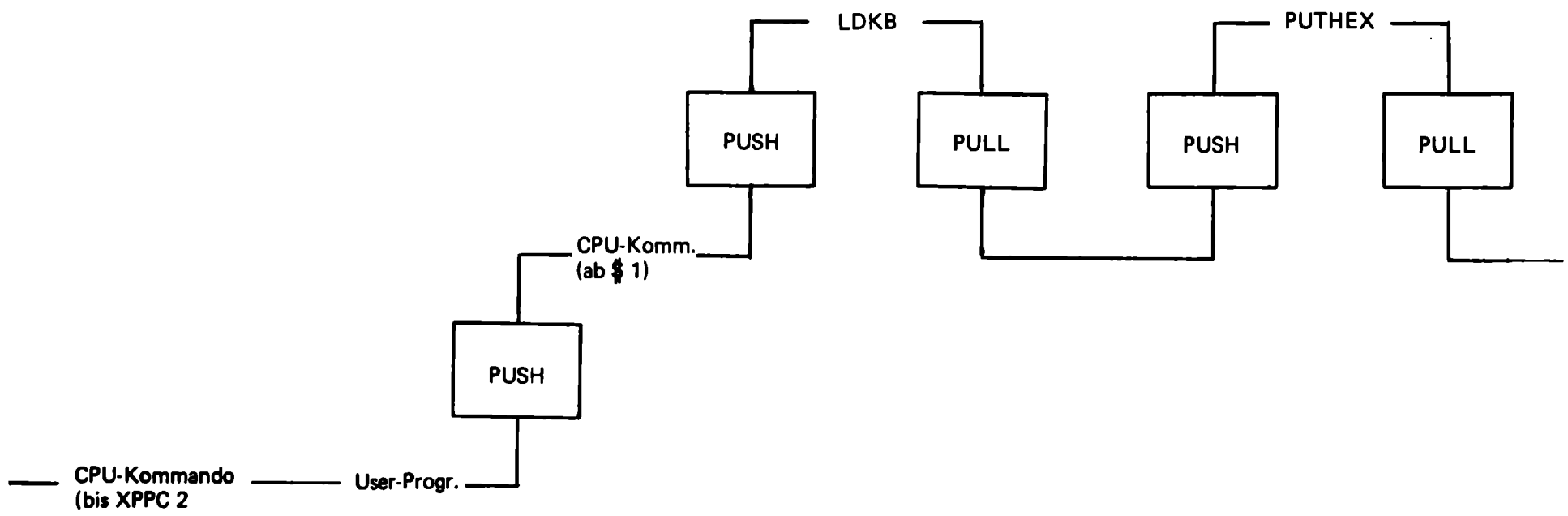


Abb. 2 Verwendung des STACK in dem CPU-Kommando



Um den betreffenden Pointer außerhalb der Stack-Routinen frei verwenden zu können, wird sein Inhalt vor Verlassen der Stack-Routinen in eine fixe RAM-Adr. geladen (STAKPT = OFFF und OFFE). In diese Adr. wird in der „ Initialize " die Adr. des Stack-Anfangs (Stack-Base = STKBSE) geschrieben.
Bei Interrupt-Verarbeitung kann u.U. eine beachtliche Stack-Tiefe zusammenkommen, die dann entweder in unbeschaltete Adressbereiche oder in RAM-Bereiche hineinläuft in welchen andere Daten oder Programme stehen. Um das zu vermeiden, wird ein Stack-Zähler (STKEFF) mitgeführt, der bei jeder Stack-Operation in- bzw. dekrementiert wird. (Updating). Außerdem ist ein über Moldify einzustellendes RAM-Byte vorgesehen (STDEEP) welches die maximal zulässige Stack-Tiefe enthält und bei jedem Status-Abspeichern mit der effektiv erreichten Tiefe (STKEFF) verglichen wird. Ist die maximale Tiefe erreicht, so wird ein „ Stack-Full "-Flag, ein RAM-Byte (= STFULL = OFFB) mit X'FF geladen.

Dieses Byte kann aus Anwender-Programmen abgefragt und falls gesetzt, ein weiterer Interrupt gesperrt werden.
Ein STATUS benötigt 11 RAM-Byte. Die Einstellung der maximal zulässigen Tiefe (STDEEP) wird für jeden kompletten STATUS um 1 erhöht.

Aus Abb. 1 ist ersichtlich daß 2 getrennte Stack-Routinen: PUSH und PULL notwendig sind. PUSH bewerkstelligt das Abspeichern der CPU-Registerinhalte in das RAM. Am Ende von PUSH befinden sich Befehle zum Ansprung der Subroutine. Deren Adresse (—1) muß vorher in die fixe RAM-Adr. „ ROUTAD " (= Adr. der Routine) geschrieben werden. Dazu bedient man sich entweder der Tatsache, daß zum Ansprung von PUSH PTR 3 mit PUSH—1 geladen werden muß und ROUTAD somit Pointer-(3)-relativ adressierbar ist (s. Listing Adr. O255) oder eines bereits vorher geladenen RAM-Pointers (s. Listing Adr. OOBF).

Hier können einige Befehlsbyte gespart werden, wenn der vorherige Inhalt von ROUTAD zum Teil identisch mit dem neuen erforderlichen Inhalt ist. Im ELBUG liegen die Subroutinen LDKB, GETHEX und PUTHEX auf der gleichen 1/4 PAGE. Das higher Adr.-Byte ist für alle 3 identisch (= Ø2). Wenn jetzt in ROUTAD noch die Adr. LDKB—1 steht und es soll GETEX angesprungen werden, so braucht das higher Byte von ROUTAD nicht mehr geladen zu werden (s. Listing Adr. O13F).

Jede Subroutine selbst muß mit JS 3 (PULL) (= Makro, s. Assembler-Manual) beendet sein, damit das Hauptprogramm (oder die Subroutine von der vorherigen „ Etage ") erst angesprungen wird, wenn der betr. STATUS aus dem STACK wieder in die CPU-Reg. geladen ist. Dieses Zurück-Laden der CPU wird durch die Stack-Routine PULL besorgt.

Falls in einer Subroutine PTR 3 nicht verwendet wird, kann sie mit XPPC 3 (anstatt „ JS 3 (PULL) ") beendet werden. PTR 3 zeigt ja dann noch auf das Ende von PUSH und hier befindet sich ein Sprung nach PULL.

Ebenso kann eine gewisse Subroutine aus einer anderen Programm-Ebene zum 2. oder jedem weiteren mal mit XPPC 3 angesprungen werden, wenn (PTR 3) inzwischen nicht verändert worden ist (s. Listing Adr. Ø37A).

Dieser befehlssparende Ansprung ist möglich, weil PULL mit einem Sprung nach PUSH beendet ist und weil (ROUTAD) genau wie die CPU-Register auch in den Stack geschoben wird (= fast ein 4. Index-Register des SC/MP).

Die hier beschriebene Handhabung der Stack-Routinen hat natürlich volle Gültigkeit auch für Anwender-Programme. Falls in diesen eine Interrupt-Verarbeitung vorgesehen ist, muß dieser solange gesperrt bleiben, bis die o.a. organisatorischen Aufgaben erledigt sind (PTR 3 auf PUSH—1 stellen und Subroutinenadresse in ROUTAD laden). Nach Freigabe wird dann das durch Interrupt ausgelöste XPPC 3 wirksam. Bei mehreren Interrupt-Eingängen muß noch eine kleine Erkennungs-Software (s. ELEKTOR) verwirklicht werden.
Zu den anderen Routinen und Kommandos ist eigentlich nicht mehr viel zu sagen. Alles wesentliche geht aus dem Listing, den Flußdiagrammen und der RAM-Tabelle hervor.
Nach PUSH stehen dem Anwender *alle* 7 Register des SC/MP zur Verfügung; sie können, je nach Bedarf neu geladen werden.

Innerhalb des ELBUG werden die Routinen bis maximal zu einer 2-fachen Verschachtelung benutzt, sodaß das RAM von Adr. OFC9 abwärts frei zur Verfügung steht. Stack-Routinen, Kommando-Schleife und alle Kommandos befinden sich auf Hauptprogramm-Ebene, die Subroutinen LDKB, GETHEX und PUTHEX auf Ebene 1 und LDKB (von GETHEX aufgerufen) auf Ebene 2. LDBYTE und BYTOUT werden nicht über den Stack angesprungen.

LDKB kann auch auf Hauptprogramm-Ebene, unter Verwendung des Ansprung-Labels LDKB1 aufgerufen werden. Der Binär-Wert der Taste wird im E-Reg. und der 7-Segm.-Code im AC „ mitgebracht ". Ein wiederholtes Anspringen mit XPPC 3 (ohne Stack) ist leider nicht möglich, weil am Ende von LDKB (aus Speicher-Platzgründen) der Befehl JMP LDKB1 fehlt.

GETHEX holt durch 4-maliges Anspringen (über den Stack) von LDKB, 4 Hex-Tasten, schreibt diese auf Display 5 bis 2, stellt die beiden Byte zusammen und speichert sie in Adr. ØFE1 — 0FE2.

PUTHEX holt 3 Byte von Adr. ØFEØ bis ØFE2, wandelt sie in 7-Segm.-Code und schreibt diesen auf Display Ø — 5. Wenn aus Anwender-Programmen Hex-Zahlen auf das Display gebracht werden sollen, geht man so vor:

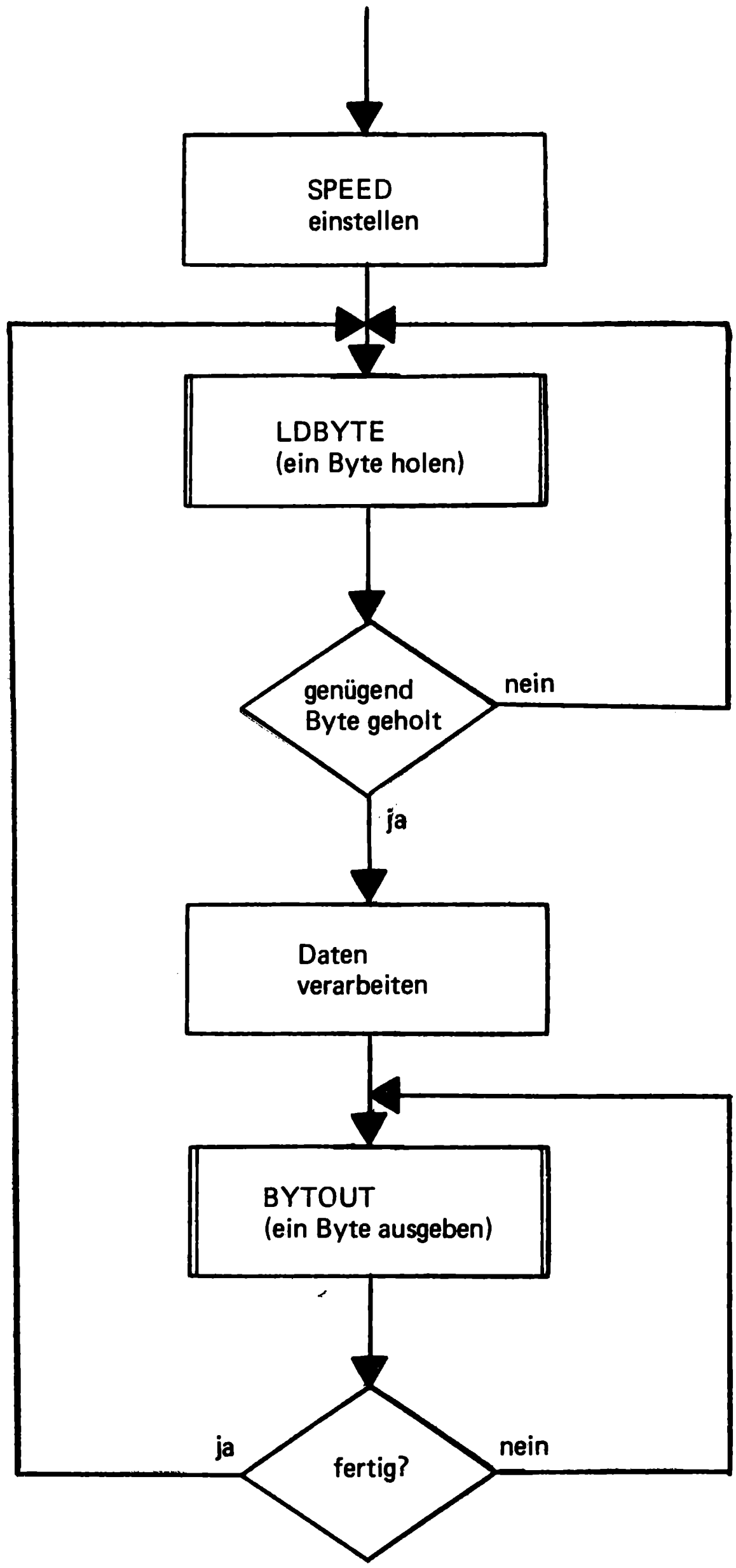
- 3 (oder weniger) Byte in Adr. ØFEØ — ØFE2 schreiben
- PTR 3 auf PUSH—1 stellen,
- PTUHEX—1 in ROUTAD schreiben (PTR-3-relativ oder über einen bereits geladenen RAM-Pointer) und
- XPPC 3.

Falls die gewünschte Zahl weniger als 6 Stellen hat, müssen anschließend die überflüssigen Display-Stellen gelöscht werden.
Die Verwendung von GETHEX geschieht genauso, nur daß die 2 Byte nach dem Rücksprung von Adr. OFE1 — OFE2 geholt werden.

In dem *CPU-KOMMANDO* ist ein kleiner Programmier-Trick enthalten. Das User-Programm wird mit XPPC 3 verlassen. PTR 3, der innerhalb dieses Programms nicht benutzt werden darf, wurde vorher auf PUSH—1 gestellt. Dadurch wird der „ STATUS " am betreffenden Punkt des User-Programms in den Stack geschoben. Anschließend springt das Programm nach „ $ 1 "-CPU-Kommando ($ 1 wurde vorher von der CPU-Kommando-Software in ROUTAD geladen).
Damit dieser „ Anwender-Status " nicht verlorengeht, wird die STACK-BASE jetzt um eine Etage „ verfälscht ". Dadurch werden die „ STATEN " der folgenden Routinen (LDKB und PTUHEX eine Etage tiefer in das RAM geschrieben und somit bleibt der „ Anwender-Status " erhalten. (Abb. 2)

Die Verwendung der Kassetten-Routinen LDBYTE und BYTOUT aus Anwenderprogrammen geschieht mit „ JS 3 (LDBYTE)" oder „ JS 3 (BYTOUT)". Ein wiederholter Ansprung mit einem einfachen XPPC 3 ist möglich.

Für ein gleichzeitiges Arbeiten mit 2 Rekordern (notwendig beim Assemblieren, Listen bearbeiten usw) kann ein eigenes Programm nach dem Flußdiagramm erstellt werden.



Verwendung von LDBYTE und BYTOUT aus User-Programmen für simultanes Arbeiten mit 2 Kassetten-Rekordern.